

EXHIBIT 2

U.S. Patent No. 7,519,814 (“’814 Patent”)

Accused Instrumentalities: Amazon products and services using secure containerized applications, including without limitation Amazon AWS Elastic Container Service (ECS) (including ECS Anywhere), AWS Elastic Kubernetes Service (EKS) (including EKS Anywhere), AWS EC2 (including spot instances), AWS Elastic Container Registry (ECR), and AWS App2Container, and all versions and variations thereof since the issuance of the asserted patent.

Each Accused Instrumentality, separately and in conjunction with each other as described below, infringes the claims in substantially the same way, and the evidence shown in this chart is similarly applicable to each Accused Instrumentality. Each claim limitation is literally infringed by each Accused Instrumentality. However, to the extent any claim limitation is not met literally, it is nonetheless met under the doctrine of equivalents because the differences between the claim limitation and each Accused Instrumentality would be insubstantial, and each Accused Instrumentality performs substantially the same function, in substantially the same way, to achieve the same result as the claimed invention. Notably, Defendant has not yet articulated which, if any, particular claim limitations it believes are not met by the Accused Instrumentalities.

Claim 1

Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a</p>	<p>To the extent the preamble is limiting, Amazon and/or its customer practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p>For example, AWS ECS, AWS EKS, and AWS EC2 each operates within a system having individual servers, each of which uses an independent operating system. Amazon provides and/or requires that each server includes a processor with one or more cores available to the OS kernel. Amazon further provides and/or requires that each server has a supported operating system (<i>e.g.</i>, Amazon Linux 2023, Amazon Linux 2, Amazon EKS Distro, Bottlerocket, Windows Server 2019, Ubuntu, Red Hat Enterprise Linux), which includes a kernel and associated local system files, including for example libraries such as libc/glibc, configuration files, etc. On information and belief, there exist at least two</p>

Claim 1	Accused Instrumentalities
<p>secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>AWS servers that have different operating systems, for example Amazon Linux 2023 and Amazon Linux 2, or different versions thereof.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <div data-bbox="640 457 1915 558" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>What is Amazon Elastic Container Service (ECS)?</p> </div> <p>Amazon ECS is a fully managed opinionated container orchestration service that delivers the easiest way for organizations to build, deploy, and manage containerized applications at any scale on AWS, in traditional Amazon Elastic Cloud Compute (EC2) instances or on a serverless compute plane with AWS Fargate. Amazon ECS is fully managed and versionless, providing tooling and built-in support that makes it simple to build and run containerized applications on AWS. For example, Amazon ECS Service Connect simplifies service discovery, connectivity, and traffic observability while Amazon CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs. With Amazon ECS, you do not have to provision or scale servers or clusters or choose the types of servers you want your containers to run on or optimize cluster packing. You retain control of the operating properties of containers with the ability to specify CPU and memory requirements, networking and IAM policies, and launch type and data volumes. With simple API calls, you can launch and stop container-enabled applications, query the complete state of your cluster, and access many familiar features like security groups, Elastic Load Balancing (ELB), Amazon Elastic Block Store (EBS) volumes, and Identity Access Management (IAM) roles. You can use Amazon ECS to schedule container placement across your cluster based on your resource needs and availability requirements.</p> <p>https://aws.amazon.com/ecs/faqs/</p>

Claim 1	Accused Instrumentalities
	<p>Q: What is Amazon Elastic Kubernetes Service (Amazon EKS)?</p> <p>A: Amazon EKS is a managed service that makes it easy for you to run Kubernetes on AWS without installing and operating your own Kubernetes control plane or worker nodes.</p> <p>Q: What is Kubernetes?</p> <p>A: Kubernetes is an open-source container orchestration system allowing you to deploy and manage containerized applications at scale. Kubernetes arranges containers into logical groupings for management and discoverability, then launches them onto clusters of Amazon Elastic Compute Cloud (Amazon EC2) instances. Using Kubernetes, you can run containerized applications including microservices, batch processing workers, and platforms as a service (PaaS) using the same toolset on premises and in the cloud.</p> <p>Q: Why should I use Amazon EKS?</p> <p>A: Amazon EKS provisions and scales the Kubernetes control plane, including the application programming interface (API) servers and backend persistence layer, across multiple AWS Availability Zones (AZs) for high availability and fault tolerance. Amazon EKS automatically detects and replaces unhealthy control plane nodes and patches the control plane. You can run EKS using AWS Fargate, which provides serverless compute for containers. Fargate removes the need to provision and manage servers, lets you specify and pay for resources per application, and improves security through application isolation by design.</p> <p>Amazon EKS is integrated with many AWS services to provide scalability and security for your applications. These services include Elastic Load Balancing for load distribution, AWS Identity and Access Management (IAM) for authentication, Amazon Virtual Private Cloud (VPC) for isolation, and AWS CloudTrail for logging.</p> <p>https://aws.amazon.com/eks/faqs/</p>

Claim 1	Accused Instrumentalities
	<p>Q: What is Amazon Elastic Compute Cloud (Amazon EC2)?</p> <p>Amazon EC2 is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers.</p> <p>Q: What can I do with Amazon EC2?</p> <p>Just as Amazon Simple Storage Service (Amazon S3) enables storage in the cloud, Amazon EC2 enables “compute” in the cloud. The Amazon EC2 simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon’s proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use.</p> <p>https://aws.amazon.com/ec2/faqs/</p> <p>Amazon Ads chose Amazon Web Services (AWS) to reduce time spent managing infrastructure, lower costs, and optimize ad selection by choosing from the broadest and deepest selection of compute and machine learning capabilities to meet its latency and performance requirements. Using Amazon Elastic Container Service (Amazon ECS) and AWS App Mesh, Amazon Ads built a micro-service inferencing architecture, which scaled model hosting and optimized hardware and software optimizations for each type of inference model. The company chose NVIDIA Triton Inference Servers running on GPU-based Amazon Elastic Compute Cloud (Amazon EC2) G4dn instances for ultra-low latency predictions with deep neural networks. For asynchronous predictions using BERT models, Amazon Ads uses Amazon SageMaker Multi-Model Endpoints running on Amazon EC2 Inf1 instances, which deliver 2.3x higher throughput and up to 70 percent lower cost per inference than comparable current generation GPU-based Amazon EC2 instances.</p> <p>https://aws.amazon.com/solutions/case-studies/amazonads-kunliu-video-case-study/?did=cr_card&trk=cr_card</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> • Container – Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. Containers share an operating system installed on the host server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of environment. A container is a runnable instance of an image (see Deep Dive on Containers on AWS getting started resource center). While a container image is immutable, the container adds a read/write layer on top of the image to which your application can write information like temporary files or logs (see Docker overview). When the container stops and is removed, the information written to the temporary read/write layer will be lost. • Container image – Container images are read-only templates used to build out containers. Container images are immutable, meaning they cannot be changed once created. Container images are created using layers by reading a text file that is called a Dockerfile that contains all necessary information. You can find the Dockerfile reference here. • Container runtime – Software running on a container host (virtual machine or bare metal server) operating system that is responsible for running and managing containers. The container relies on the kernel of the host for all system calls. • Dockerfile – A file or series of files containing commands that describe the content of a container image. Each command represents a layer on the Container image (see the <i>Container image layers</i> definition). • Container build – A container image built from a Dockerfile. This process results in a container image containing the necessary components to run your containerized application. • Base image – A starting point container image that is used in the container build process to generate custom or new container images. This image has <code>FROM scratch</code> in the Dockerfile. <p>https://docs.aws.amazon.com/wellarchitected/latest/container-build-lens/container-technology-terminology.html</p>

Claim 1	Accused Instrumentalities
	<p>Container technology uses the resource-isolation features of the Linux kernel to sandbox an application, its dependencies, configuration files, and interfaces inside an atomic unit called a container. This allows a container to run on any host with the suitable kernel components, while shielding the application from behavioral inconsistencies through variances in software installed on the host. Containers use operating system (OS) level virtualization compared to VMs, which use hardware level virtualization using hypervisor. A hypervisor is a software or a firmware that creates and runs VMs. Multiple containers can run on a single host OS without needing a hypervisor, while isolated from neighboring containers. This layer of isolation allows consistency, flexibility, and portability, which enable rapid software deployment and testing. There are many ways in which using containers on AWS can benefit your organization. Containers have been widely employed in use cases such as distributed applications, batch jobs, and continuous deployment pipelines. The use cases for containers continue to grow in areas like distributed data processing, streaming media delivery, genomics, and machine learning, including generative AI.</p> <p>https://docs.aws.amazon.com/whitepapers/latest/containers-on-aws/containers-on-aws.html</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="667 248 1577 305">Amazon ECS-optimized Linux AMIs</h2> <div data-bbox="667 332 789 358">PDF RSS</div> <p data-bbox="667 423 1902 769">Amazon ECS provides the Amazon ECS-optimized AMIs that are preconfigured with the requirements and recommendations to run your container workloads. We recommend that you use the Amazon ECS-optimized Amazon Linux 2023 AMI for your Amazon EC2 instances unless your application requires Amazon EC2 GPU-based instances, a specific operating system or a Docker version that is not yet available in that AMI. For information about the Amazon Linux 2 and Amazon Linux 2023 instances, see Comparing Amazon Linux 2 and Amazon Linux 2023 in the <i>Amazon Linux 2023 User Guide</i>. Launching your container instances from the most recent Amazon ECS-Optimized AMI ensures that you receive the current security updates and container agent version. For information about how to launch an instance, see Launching an Amazon ECS Linux container instance.</p> <p data-bbox="632 800 1913 902">https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-optimized_AMI.html; <i>see also</i> list of differences between Amazon Linux 2 and Amazon Linux 2023 at https://docs.aws.amazon.com/linux/al2023/ug/compare-with-al2.html</p>

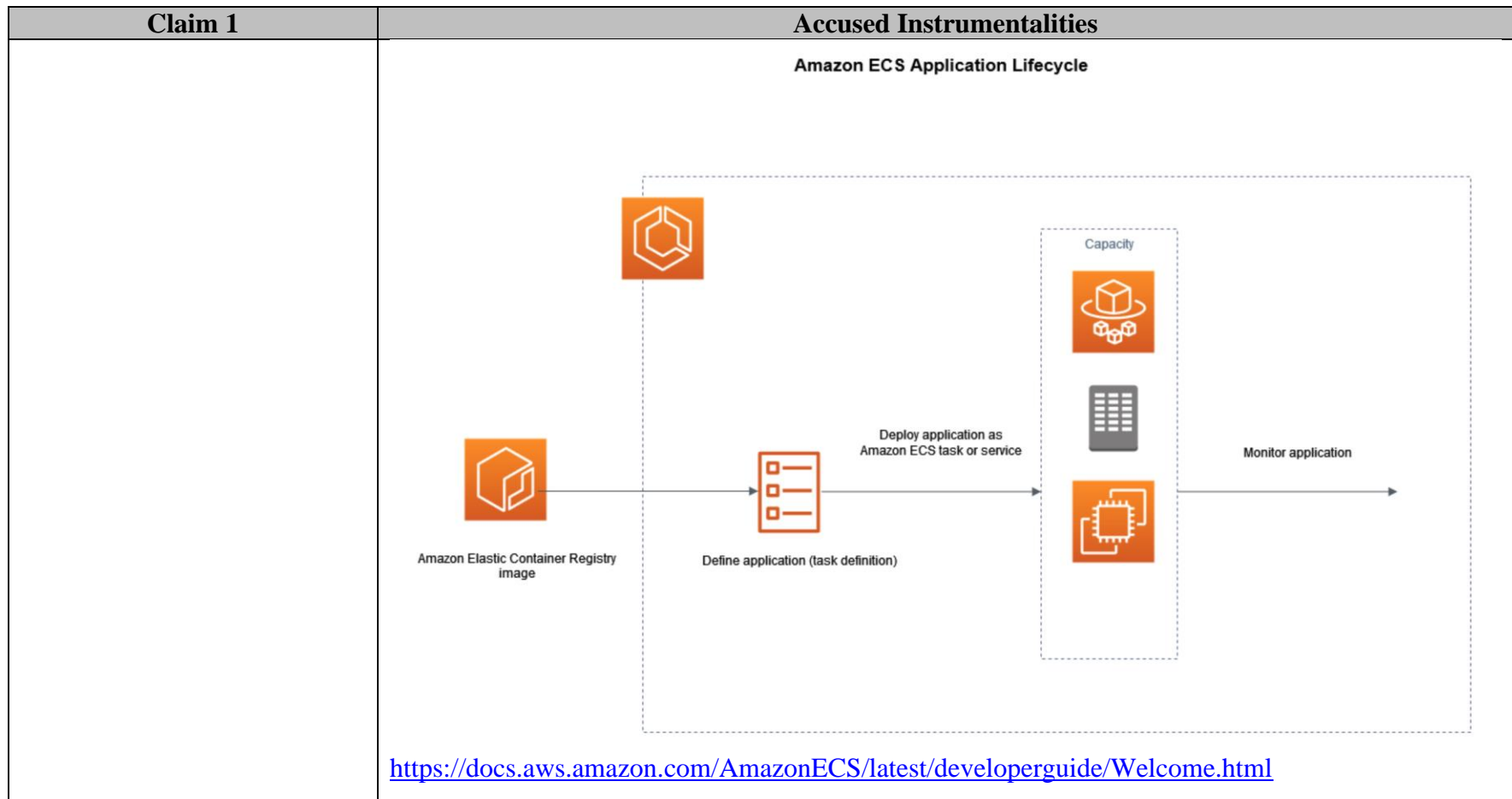
Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 256 1749 318">Amazon ECS-optimized Bottlerocket AMIs</h2> <p data-bbox="659 342 779 370"> PDF RSS </p> <p data-bbox="659 435 1890 662"> Bottlerocket is a Linux based open-source operating system that is purpose built by AWS for running containers on virtual machines or bare metal hosts. The Amazon ECS-optimized Bottlerocket AMI is secure and only includes the minimum number of packages that's required to run containers. This improves resource usage, reduces security attack surface, and helps lower management overhead. The Bottlerocket AMI is also integrated with Amazon ECS to help reduce the operational overhead involved in updating container instances in a cluster. </p> <p data-bbox="632 686 1743 714"> https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-bottlerocket.html </p> <h2 data-bbox="669 760 1671 821">Amazon ECS-optimized Windows AMIs</h2> <p data-bbox="669 846 789 873"> PDF RSS </p> <p data-bbox="669 938 1881 1122"> The Amazon ECS-optimized AMIs are preconfigured with the necessary components that you need to run Amazon ECS workloads. Although you can create your own container instance AMI that meets the basic specifications needed to run your containerized workloads on Amazon ECS, the Amazon ECS-optimized AMIs are preconfigured and tested on Amazon ECS by AWS engineers. It is the simplest way for you to get started and to get your containers running on AWS quickly. </p> <p data-bbox="632 1146 1927 1182"> https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-optimized_windows_AMI.html </p>

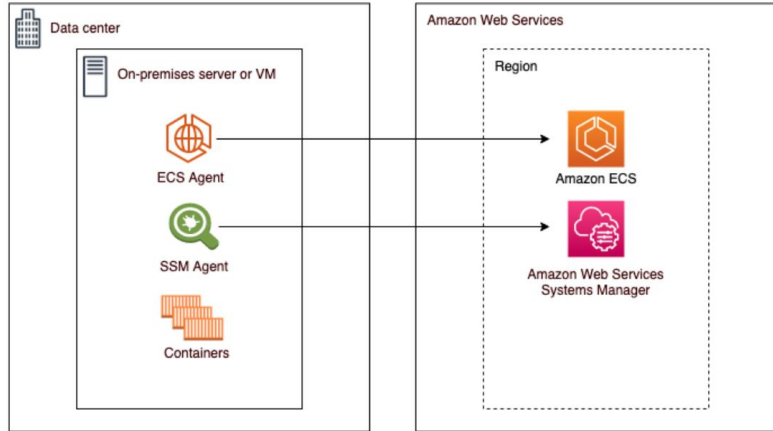
Claim 1	Accused Instrumentalities
	<h2 data-bbox="667 272 1812 332">Amazon EKS optimized Amazon Linux AMIs</h2> <p data-bbox="667 358 785 386">PDF RSS</p> <p data-bbox="667 451 1873 560">The Amazon EKS optimized Amazon Linux AMI is built on top of Amazon Linux 2 (AL2) and Amazon Linux 2023 (AL2023). It's configured to serve as the base image for Amazon EKS nodes. The AMI is configured to work with Amazon EKS and it includes the following components:</p> <ul data-bbox="682 597 1289 771" style="list-style-type: none"> • kubelet • AWS IAM Authenticator • Docker (Amazon EKS version 1.23 and earlier) • containerd <p data-bbox="636 800 1581 828">https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-ami.html</p> <h2 data-bbox="674 857 1745 917">Amazon EKS optimized Bottlerocket AMIs</h2> <p data-bbox="674 943 791 971">PDF RSS</p> <p data-bbox="674 1031 1898 1295">Bottlerocket is an open source Linux distribution that's sponsored and supported by AWS. Bottlerocket is purpose-built for hosting container workloads. With Bottlerocket, you can improve the availability of containerized deployments and reduce operational costs by automating updates to your container infrastructure. Bottlerocket includes only the essential software to run containers, which improves resource usage, reduces security threats, and lowers management overhead. The Bottlerocket AMI includes containerd, kubelet, and AWS IAM Authenticator. In addition to managed node groups and self-managed nodes, Bottlerocket is also supported by Karpenter.</p> <p data-bbox="636 1333 1745 1360">https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-ami-bottlerocket.html</p>


Claim 1	Accused Instrumentalities
	<h2 data-bbox="667 256 1734 313">Amazon EKS optimized Ubuntu Linux AMIs</h2> <div data-bbox="667 337 783 365">PDF RSS</div> <p data-bbox="667 423 1776 451">Canonical has partnered with Amazon EKS to create node AMIs that you can use in your clusters.</p> <p data-bbox="667 487 1856 667">Canonical delivers a built-for-purpose Kubernetes Node OS image. This minimized Ubuntu image is optimized for Amazon EKS and includes the custom AWS kernel that is jointly developed with AWS. For more information, see Ubuntu on Amazon Elastic Kubernetes Service (EKS) and Launching self-managed Ubuntu nodes. For information about support, see the Third-party software section of the <i>AWS Premium Support FAQs</i>.</p> <p data-bbox="632 695 1556 722">https://docs.aws.amazon.com/eks/latest/userguide/eks-partner-amis.html</p>

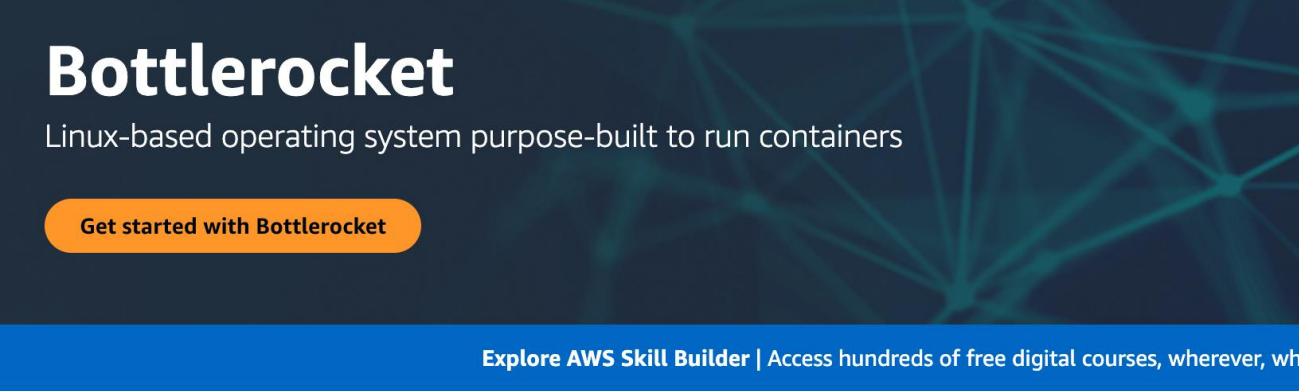
Claim 1	Accused Instrumentalities
	<h2 data-bbox="667 261 1667 321">Amazon EKS optimized Windows AMIs</h2> <p data-bbox="667 347 785 375"> PDF RSS </p> <p data-bbox="667 436 1877 545">Windows Amazon EKS optimized AMIs are built on top of Windows Server 2019 and Windows Server 2022. They are configured to serve as the base image for Amazon EKS nodes. By default, the AMIs include the following components:</p> <ul data-bbox="684 578 1213 802" style="list-style-type: none"> • kubelet • kube-proxy • AWS IAM Authenticator for Kubernetes • csi-proxy • containerd <div data-bbox="667 834 1902 1015"> <p>Note</p> <p>You can track security or privacy events for Windows Server with the Microsoft security update guide.</p> </div> <p data-bbox="667 1047 1785 1075">Amazon EKS offers AMIs that are optimized for Windows containers in the following variants:</p> <ul data-bbox="684 1115 1383 1287" style="list-style-type: none"> • Amazon EKS-optimized Windows Server 2019 Core AMI • Amazon EKS-optimized Windows Server 2019 Full AMI • Amazon EKS-optimized Windows Server 2022 Core AMI • Amazon EKS-optimized Windows Server 2022 Full AMI <p data-bbox="634 1308 1707 1341"> https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-windows-ami.html </p>

Claim 1	Accused Instrumentalities
	<p>Q: Which operating systems does Amazon EKS support?</p> <p>A: Amazon EKS supports Kubernetes-compatible Linux x86, ARM, and Windows Server operating system distributions. Amazon EKS provides optimized AMIs for Amazon Linux 2, Bottlerocket, and Windows Server 2019. At this time, there is no Amazon EKS optimized AMI for AL2023. EKS- optimized AMIs for other Linux distributions, such as Ubuntu, are available from their respective vendors.</p> <p>https://aws.amazon.com/eks/faqs/</p> <p>Q: What infrastructure and operating systems can I use with Amazon EKS Anywhere?</p> <p>Amazon EKS Anywhere supports different types of infrastructure including VMWare vSphere, bare metal, AWS Snowball Edge, Apache CloudStack, and Nutanix. Amazon EKS Anywhere provides Bottlerocket, a Linux-based open-source operating system built by AWS, as the default node operating system. You can alternatively use Ubuntu and Red Hat Enterprise Linux (RHEL) as the node operating system.</p> <p>https://aws.amazon.com/eks/eks-anywhere/faqs/</p>



Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 250 1285 367">Amazon ECS clusters for the external launch type</h2> <p data-bbox="646 388 751 410"> PDF RSS </p> <p data-bbox="646 466 1434 727"> Amazon ECS Anywhere provides support for registering an <i>external instance</i> such as an on-premises server or virtual machine (VM), to your Amazon ECS cluster. External instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, the lack of Elastic Load Balancing support makes running these workloads less efficient. Amazon ECS added a new <code>EXTERNAL</code> launch type that you can use to create services or run tasks on your external instances. </p> <p data-bbox="646 761 1442 854"> The following provides a high-level system architecture overview of Amazon ECS Anywhere. Your on-premises server has both the Amazon ECS agent and the SSM agent installed. </p>  <p data-bbox="632 1365 1715 1398"> https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-anywhere.html </p>

Claim 1	Accused Instrumentalities
	<p data-bbox="646 235 1375 267">Supported operating systems and system architectures</p> <p data-bbox="646 292 1289 313">The following is the list of supported operating systems and system architectures.</p> <ul data-bbox="657 337 1898 570" style="list-style-type: none"> • Amazon Linux 2 • CentOS 7 • RHEL 7, RHEL 8 — Neither Docker or RHEL's open package repositories support installing Docker natively on RHEL. You must ensure that Docker is installed before you run the install script that's described in this document. • Fedora 32, Fedora 33 • openSUSE Tumbleweed • Ubuntu 18, Ubuntu 20, Ubuntu 22 • Debian 10 <div data-bbox="680 591 1915 683" style="border: 1px solid #f08080; padding: 10px; margin: 10px 0;"> <p data-bbox="701 610 816 631"> Important</p> <p data-bbox="730 643 1692 664">Debian 9 Long Term Support (LTS support) ended on June 30, 2022 and is no longer supported by Amazon ECS Anywhere.</p> </div> <ul data-bbox="657 711 1761 980" style="list-style-type: none"> • Debian 11 • Debian 12 — The NVIDIA Container Toolkit isn't currently supported on Debian 12. You won't be able to run GPUs on Debian 12 instances. • SUSE Enterprise Server 15 • The <code>x86_64</code> and <code>ARM64</code> CPU architectures are supported. • The following Windows operating system versions are supported: <ul data-bbox="688 867 884 980" style="list-style-type: none"> ◦ Windows Server 2022 ◦ Windows Server 2019 ◦ Windows Server 2016 ◦ Windows Server 20H2 <p data-bbox="636 1005 1713 1037">https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-anywhere.html</p>

Claim 1	Accused Instrumentalities
	 <p>Bottlerocket is a Linux-based open-source operating system that is purpose-built by Amazon Web Services for running containers. Bottlerocket includes only the essential software required to run containers, and ensures that the underlying software is always secure. With Bottlerocket, customers can reduce maintenance overhead and automate their workflows by applying configuration settings consistently as nodes are upgraded or replaced.</p> <p>Bottlerocket is now generally available at no cost as an Amazon Machine Image (AMI) for Amazon Elastic Compute Cloud (EC2).</p> <p>https://aws.amazon.com/bottlerocket/</p>
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a	<p>The method practiced by Amazon and/or its customer through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p>For example, AWS ECS, AWS EKS, and AWS EC2 each stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent</p>

Claim 1	Accused Instrumentalities
<p>set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>storage available to each node running the application. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network, either within the AWS environment or on-premises. Each container includes the application software as well as a Linux or Windows user space required to execute the application, for example libc/glibc and other shared libraries (or Windows equivalents, such as DLLs), configuration files, etc. necessary for the application. For example, the container includes a base OS image, provided by Amazon or by a third party, such as an Amazon Linux, Alpine Linux, Debian, Ubuntu, or Windows Server base image. The container is compatible with the host kernel, for example because the container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface. Likewise, Windows applications are compiled to work with Windows, which has a stable binary interface.</p> <p>For another example, AWS ECS, AWS EKS, and AWS EC2 each stores files, pertaining to the applications, in ephemeral or persistent volumes, required to execute the applications within those containers. Because these volumes are stored and accessible within the runtime environment, it is inferred that they are stored in the memory of the server as claimed.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="674 248 1780 310">Architect your application for Amazon ECS</h2> <p data-bbox="674 334 793 358"> PDF RSS </p> <p data-bbox="674 427 1860 493"> You architect your application by creating a task definition for your application. The task definition contains the parameters that define information about the application, including: </p> <ul data-bbox="695 532 1885 824" style="list-style-type: none"> <li data-bbox="695 532 1885 688"> • The launch type to use, which determines the infrastructure that your tasks are hosted on. When you use the EC2 launch type, you also choose the instance type. For some instance types, such as GPU, you need to set additional parameters. For more information, see Amazon ECS task definition use cases. <li data-bbox="695 711 1818 777"> • The container image, which holds your application code and all the dependencies that your application code requires to run. <li data-bbox="695 800 1885 1040"> • The networking mode to use for the containers in your task The networking mode determines how your task communicates over the network. For tasks that run on EC2 instance, there are multiple options, but we recommend that you use the <code>awsvpc</code> network mode. The <code>awsvpc</code> network mode simplifies container networking, because you have more control over how your applications communicate with each other and other services within your VPCs. For tasks that run on Fargate, you can only use the <code>awsvpc</code> network mode. <li data-bbox="695 1110 1297 1136"> • The logging configuration to use for your tasks. <li data-bbox="695 1159 1482 1185"> • Any data volumes that are used with the containers in the task. <p data-bbox="636 1208 1848 1234"> https://docs.aws.amazon.com/AmazonECS/latest/developerguide/application_architecture.html </p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="667 256 1213 316">Container image type</h2> <p data-bbox="667 337 783 365"> PDF RSS </p> <p data-bbox="667 427 1843 573"> The time that it takes a container to start up varies, based on the underlying container image. For example, a fatter image (full versions of Debian, Ubuntu, and Amazon1/2) might take longer to start up because there are a more services that run in the containers compared to their respective slim versions (Debian-slim, Ubuntu-slim, and Amazon-slim) or smaller base images (Alpine). </p> <p data-bbox="634 625 1764 657"> https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/container-type.html </p> <ul data-bbox="655 695 1911 1320" style="list-style-type: none"> • Container – Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. Containers share an operating system installed on the host server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of environment. A container is a runnable instance of an image (see Deep Dive on Containers on AWS getting started resource center). While a container image is immutable, the container adds a read/write layer on top of the image to which your application can write information like temporary files or logs (see Docker overview). When the container stops and is removed, the information written to the temporary read/write layer will be lost. • Container image – Container images are read-only templates used to build out containers. Container images are immutable, meaning they cannot be changed once created. Container images are created using layers by reading a text file that is called a Dockerfile that contains all necessary information. You can find the Dockerfile reference here. • Container runtime – Software running on a container host (virtual machine or bare metal server) operating system that is responsible for running and managing containers. The container relies on the kernel of the host for all system calls. • Dockerfile – A file or series of files containing commands that describe the content of a container image. Each command represents a layer on the Container image (see the <i>Container image layers</i> definition). • Container build – A container image built from a Dockerfile. This process results in a container image containing the necessary components to run your containerized application. • Base image – A starting point container image that is used in the container build process to generate custom or new container images. This image has FROM scratch in the Dockerfile. <p data-bbox="634 1328 1843 1401"> https://docs.aws.amazon.com/wellarchitected/latest/container-build-lens/container-technology-terminology.html </p>

Claim 1	Accused Instrumentalities
	<p>Base image</p> <p>The base image is the selected image and operating system used in your image or container recipe document, along with the components. The base image and the component definitions combined produce the desired configuration for the output image.</p> <p>https://docs.aws.amazon.com/imagebuilder/latest/userguide/what-is-image-builder.html</p> <p>Using a base image from a trusted source can improve the security and reliability of your container. This is because you can be confident that the base image has been thoroughly tested and vetted. It can also reduce the burden of establishing provenance, because you only need to consider the packages and libraries that you include in your image, rather than the entire base image. Here is an example creating a Dockerfile using the official Python base image from the Amazon ECR repository. In fact, all of the Docker official images are available on Amazon ECR public gallery.</p> <p>https://aws.amazon.com/blogs/containers/building-better-container-images/</p>

Claim 1	Accused Instrumentalities
	<p>Docker Official Images now available on Amazon Elastic Container Registry Public</p> <p>by Saleem Muhammad on 29 NOV 2021 in Amazon Elastic Container Registry, Announcements, Containers Permalink </p> <p>Developers building container-based applications can now discover and download Docker Official Images directly from Amazon Elastic Container Registry (Amazon ECR) Public. This new capability gives AWS customers a simple and highly available way to pull Docker Official Images, while taking advantage of the generous AWS Free Tier. Customers pulling images from Amazon ECR Public to any AWS Region get virtually unlimited downloads. For workloads running outside of AWS, users not authenticated on AWS receive 500 GB of data downloads each month. For additional data downloads, they can sign up or sign in to an AWS account to get up to 5TB of data downloads each month after which they pay \$0.09 per GB.</p> <p>Docker Official Images are a curated set of container images published by Docker. Some examples of these images include base OS (for example Ubuntu and CentOS), databases (for example MySQL and Redis), and sidecars that are the starting point for container-based applications. Until today, customers using Docker Official Images could only find these images on Docker Hub, a container registry hosted by Docker.</p> <p>https://aws.amazon.com/blogs/containers/docker-official-images-now-available-on-amazon-elastic-container-registry-public/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="669 256 1110 298">Create a Docker image</h2> <p data-bbox="669 339 1871 483">Amazon ECS task definitions use Docker images to launch containers on the container instances in your clusters. In this section, you create a Docker image of a simple web application, and test it on your local system or Amazon EC2 instance, and then push the image to the Amazon ECR container registry so you can use it in an Amazon ECS task definition.</p> <p data-bbox="669 521 1314 548">To create a Docker image of a simple web application</p> <ol data-bbox="684 586 1864 691" style="list-style-type: none"> 1. Create a file called <code>Dockerfile</code>. A Dockerfile is a manifest that describes the base image to use for your Docker image and what you want installed and running on it. For more information about Dockerfiles, go to the Dockerfile Reference. <p data-bbox="636 730 1833 760">https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-container-image.html</p> <h2 data-bbox="695 818 1709 876">Using the AL2023 base container image</h2> <p data-bbox="695 902 810 927">PDF RSS</p> <p data-bbox="695 989 1887 1133">The AL2023 container image is built from the same software components that are included in the AL2023 AMI. It's available for use in any environment as a base image for Docker workloads. If you're using the Amazon Linux AMI for applications in Amazon Elastic Compute Cloud (Amazon EC2), you can containerize your applications with the Amazon Linux container image.</p> <p data-bbox="636 1161 1478 1190">https://docs.aws.amazon.com/linux/al2023/ug/base-container.html</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="688 245 1751 305">Pulling the Amazon Linux container image</h2> <div data-bbox="688 326 806 354">PDF RSS</div> <p data-bbox="688 412 1877 558">The Amazon Linux container image is built from the same software components that are included in the Amazon Linux AMI. The Amazon Linux container image is available for use in any environment as a base image for Docker workloads. If you use the Amazon Linux AMI for applications in Amazon EC2, you can containerize your applications with the Amazon Linux container image.</p> <p data-bbox="688 591 1877 699">You can use the Amazon Linux container image in your local development environment and then push your application to AWS using Amazon ECS. For more information, see Using Amazon ECR images with Amazon ECS.</p> <p data-bbox="632 716 1877 748">https://docs.aws.amazon.com/AmazonECR/latest/userguide/amazon_linux_container_image.html</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 250 1190 310">Store application data</h2> <p data-bbox="653 331 764 358">PDF RSS</p> <p data-bbox="653 412 1339 440">This chapter covers storage options for Amazon EKS clusters.</p> <p data-bbox="653 477 730 505">Topics</p> <ul data-bbox="653 537 1199 883" style="list-style-type: none"> • Use Amazon EBS storage • Use Amazon EFS storage • Use Amazon FSx for Lustre storage • Use Amazon FSx for NetApp ONTAP storage • Use Amazon FSx for OpenZFS storage • Use Amazon File Cache • Use Mountpoint for Amazon S3 storage • Use snapshot controller with CSI storage <p data-bbox="632 920 1436 954">https://docs.aws.amazon.com/eks/latest/userguide/storage.html</p> <h3 data-bbox="678 997 1281 1036">Persistent storage for Kubernetes</h3> <p data-bbox="678 1045 1850 1105">by Suman Debnath, Daniel Rubinstein, Anjani Reddy, and Narayana Vemburaj on 22 NOV 2022 in Advanced (300), Amazon Elastic File System (EFS), Amazon Elastic Kubernetes Service, Technical How-to Permalink Comments </p> <p data-bbox="678 1164 1898 1263">Stateful applications rely on data being persisted and retrieved to run properly. When running stateful applications using Kubernetes, state needs to be persisted regardless of container, pod, or node crashes or terminations. This requires persistent storage, that is, storage that lives beyond the lifetime of the container, pod, or node.</p> <p data-bbox="632 1284 1560 1318">https://aws.amazon.com/blogs/storage/persistent-storage-for-kubernetes/</p>

Claim 1	Accused Instrumentalities
	<p>Kubernetes volumes</p> <p>Kubernetes has several types of storage options available, not all of which are persistent.</p> <p>Ephemeral storage</p> <p>Containers can use the <code>temporary filesystem</code> (tmpfs) to read and write files. However, ephemeral storage does not satisfy the three storage requirements. In case of a container crash, the <code>temporary filesystem</code> is lost—the container starts with a clean slate again. Also, multiple containers cannot share a <code>temporary filesystem</code>.</p> <p>Ephemeral volumes</p> <p>An ephemeral Kubernetes <code>Volume</code> solves both of the problems faced with ephemeral storage. An ephemeral <code>Volume</code>'s lifetime is coupled to the <code>Pod</code>. It enables safe container restarts and sharing of data between containers within a <code>Pod</code>. However as soon as the <code>Pod</code> is deleted, the <code>Volume</code> is deleted as well, so it still does not fulfill our three requirements.</p> <div data-bbox="667 782 1751 1222"> <p>The diagram illustrates two storage scenarios within a Pod. On the left, a single container has its own 'Temporary filesystem' (represented by a cylinder icon) and a 'Mount path' of '/usr/u1/nginx/html'. On the right, a Pod contains two containers, 'Container 1' and 'Container 2'. Both containers share a common 'Volume' (represented by an orange cylinder icon) located outside the containers but within the Pod. Each container has a 'Mount path' of '/var/logs' that points to this shared volume. Labels with arrows identify the 'Pod', 'Container', 'Tmp filesystem', 'Mount path', 'Temporary filesystem', 'Volume', and 'Mount path' components.</p> </div> <p><i>The temporary file system is tied to the lifecycle of the container; the ephemeral Volume is tied to the lifecycle of the pod</i></p> <p>https://aws.amazon.com/blogs/storage/persistent-storage-for-kubernetes/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="667 250 1482 289">Decoupling pods from the storage: Persistent Volumes</p> <p data-bbox="667 326 1854 431">Kubernetes also supports Persistent Volumes . With Persistent Volumes , data is persisted regardless of the lifecycle of the application, container, Pod, Node, or even the cluster itself. Persistent Volumes fulfill the three requirements outlined earlier.</p> <p data-bbox="667 472 1871 537">A Persistent Volume (PV) object represents a storage volume that is used to persist application data. A PV has its own lifecycle, separate from the lifecycle of Kubernetes Pods .</p> <p data-bbox="667 578 1163 610">A PV essentially consists of two different things:</p> <ul data-bbox="680 643 1470 724" style="list-style-type: none"> • A backend technology called a PersistentVolume • An access mode, which tells Kubernetes how the volume should be mounted. <p data-bbox="667 756 905 789">Backend technology</p> <p data-bbox="667 821 1864 854">A PV is an abstract component, and the actual physical storage must come from somewhere. Here are a few examples:</p> <ul data-bbox="680 886 1719 1081" style="list-style-type: none"> • csi : Container Storage Interface (CSI) → (for example, Amazon EFS, Amazon EBS, Amazon FSx, etc.) • iscsi : iSCSI (SCSI over IP) storage • local : Local storage devices mounted on nodes • nfs : Network File System (NFS) storage <p data-bbox="634 1097 1560 1130">https://aws.amazon.com/blogs/storage/persistent-storage-for-kubernetes/</p>

Claim 1	Accused Instrumentalities
	<p>Container Storage Interface (CSI) drivers</p> <p>The Container Storage Interface (CSI) is an abstraction designed to facilitate using different storage solutions with Kubernetes. Different storage vendors can develop their own drivers that implement the CSI standards, enabling their storage solutions to work with Kubernetes (regardless of the internals of the underlying storage solution). AWS has CSI plugins for Amazon EBS, Amazon EFS, and Amazon FSx for Lustre.</p> <p>https://aws.amazon.com/blogs/storage/persistent-storage-for-kubernetes/</p> <p>6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p>Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p>https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p> <p>At its core, a volume is a directory, possibly with some data in it, which is accessible to the containers in a pod. How that directory comes to be, the</p> <p><code>.spec.containers[*].volumeMounts</code>. A process in a container sees a filesystem view composed from the initial contents of the <u>container image</u>, plus volumes (if defined) mounted inside the container. The process sees a root filesystem that initially matches the contents of the container image. Any writes to within that filesystem hierarchy, if allowed, affect what that process views when it performs a subsequent filesystem access. Volumes mount at the specified paths within the image. For each container defined within a Pod, you must independently specify where to mount each volume that the container uses.</p> <p>https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="653 269 1150 310">What is containerization?</p> <p data-bbox="653 345 1896 505">Containerization is a software deployment process that bundles an application's code with all the files and libraries it needs to run on any infrastructure. Traditionally, to run any application on your computer, you had to install the version that matched your machine's operating system. For example, you needed to install the Windows version of a software package on a Windows machine. However, with containerization, you can create a single software package, or <a data-bbox="1377 448 1472 472" href="https://aws.amazon.com/what-is/containerization/">container, that runs on all types of devices and operating systems.</p> <p data-bbox="634 529 1268 561">https://aws.amazon.com/what-is/containerization/</p> <p data-bbox="653 597 1310 638">How does containerization work?</p> <p data-bbox="653 673 1911 833">Containerization involves building self-sufficient software packages that perform consistently, regardless of the machines they run on. Software developers create and deploy container images—that is, files that contain the necessary information to run a containerized application. Developers use containerization tools to build container images based on the Open Container Initiative (OCI) image specification. OCI is an open-source group that provides a standardized format for creating container images. Container images are read-only and cannot be altered by the computer system.</p> <p data-bbox="653 873 1579 898">Container images are the top layer in a containerized system that consists of the following layers.</p> <p data-bbox="634 914 1268 946">https://aws.amazon.com/what-is/containerization/</p>

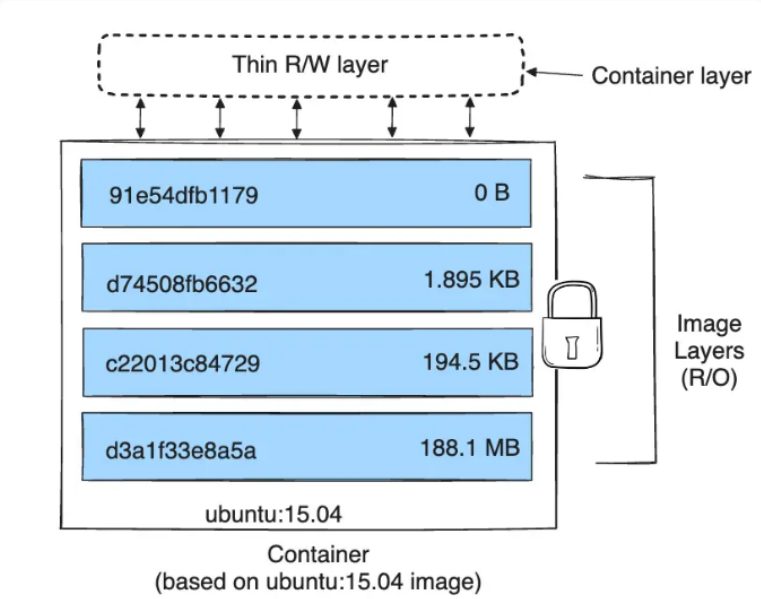
Claim 1	Accused Instrumentalities
	<p>Infrastructure</p> <p>Infrastructure is the hardware layer of the container model. It refers to the physical computer or bare-metal server that runs the containerized application.</p> <p>Operating system</p> <p>The second layer of the containerization architecture is the operating system. Linux is a popular operating system for containerization with on-premise computers. In cloud computing, developers use cloud services such as AWS EC2 to run containerized applications.</p> <p>Container engine</p> <p>The container engine, or container runtime, is a software program that creates containers based on the container images. It acts as an intermediary agent between the containers and the operating system, providing and managing resources that the application needs. For example, container engines can manage multiple containers on the same operating system by keeping them independent of the underlying infrastructure and each other.</p> <p>Application and dependencies</p> <p>The topmost layer of the containerization architecture is the application code and the other files it needs to run, such as library dependencies and related configuration files. This layer might also contain a light guest operating system that gets installed over the host operating system.</p> <p>https://aws.amazon.com/what-is/containerization/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="646 253 1528 302">What are the types of container technology?</p> <p data-bbox="646 329 1593 354">The following are some examples of popular technologies that developers use for containerization.</p> <p data-bbox="646 391 730 415">Docker</p> <p data-bbox="646 453 1913 545">Docker, or Docker Engine, is a popular open-source container runtime that allows software developers to build, deploy, and test containerized applications on various platforms. Docker containers are self-contained packages of applications and related files that are created with the Docker framework.</p> <p data-bbox="646 583 714 607">Linux</p> <p data-bbox="646 644 1904 769">Linux is an open-source operating system with built-in container technology. Linux containers are self-contained environments that allow multiple Linux-based applications to run on a single host machine. Software developers use Linux containers to deploy applications that write or read large amounts of data. Linux containers do not copy the entire operating system to their virtualized environment. Instead, the containers consist of necessary functionalities allocated in the Linux namespace.</p> <p data-bbox="646 807 781 831">Kubernetes</p> <p data-bbox="646 868 1883 961">Kubernetes is a popular open-source container orchestrator that software developers use to deploy, scale, and manage a vast number of microservices. It has a declarative model that makes automating containers easier. The declarative model ensures that Kubernetes takes the appropriate action to fulfil the requirements based on the configuration files.</p> <p data-bbox="632 989 1268 1021">https://aws.amazon.com/what-is/containerization/</p>

Claim 1	Accused Instrumentalities
	<p>How Docker containers work</p> <p>A Docker container is a runtime environment with all the necessary components—like code, dependencies, and libraries—needed to run the application code without using host machine dependencies. This container runtime runs on the engine on a server, machine, or cloud instance. The engine runs multiple containers depending on the underlying resources available.</p> <p>To deploy and scale a set of containers to communicate effectively across different machines or virtual machines, you need a container orchestration platform like Kubernetes. This helps whether your machines are on premises or in the cloud. Kubernetes manages multiple machines, known as a cluster, within the context of container operations.</p> <p>Read about Kubernetes »</p> <p>How Docker images work</p> <p>A Docker image, or container image, is a standalone, executable file used to create a container. This container image contains all the libraries, dependencies, and files that the container needs to run. A Docker image is shareable and portable, so you can deploy the same image in multiple locations at once—much like a software binary file.</p> <p>You can store images in registries to keep track of complex software architectures, projects, business segments, and user group access. For instance, the public Docker Hub registry contains images such as operating systems, programming language frameworks, databases, and code editors.</p> <p>https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 248 1272 316">About storage drivers</h2> <p data-bbox="646 363 1871 488">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 557 1564 613">Storage drivers versus Docker volumes</h2> <p data-bbox="646 651 1913 919">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 967 1902 1092">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1015 1535 1040" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1122 1224 1154"><a data-bbox="636 1122 1224 1154" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 245 1081 302">Images and layers</h2> <p data-bbox="657 337 1822 415">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 483 1451 797"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 862 1900 1170">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="632 1192 1224 1224">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a Docker container. At the top, a dashed box labeled "Thin R/W layer" (also labeled "Container layer" with an arrow) represents the writable layer for the container. Below this, a stack of four solid blue boxes represents the "Image Layers (R/O)". These layers are labeled with their commit IDs and sizes: "91e54dfb1179" (0 B), "d74508fb6632" (1.895 KB), "c22013c84729" (194.5 KB), and "d3a1f33e8a5a" (188.1 MB). A padlock icon is shown next to the stack, indicating they are read-only. The entire stack is labeled "ubuntu:15.04". Below the stack, the text "Container (based on ubuntu:15.04 image)" is displayed. Vertical double-headed arrows connect the "Thin R/W layer" to each of the four image layers, showing the relationship between the container's state and the underlying image layers.</p> <p>https://docs.docker.com/storage/storagedriver/</p>




Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 256 919 321">Volumes</h2> <p data-bbox="653 375 1906 505">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="634 529 1308 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="653 610 1226 659">Container environment</h2> <p data-bbox="653 699 1474 764">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="695 805 1451 959" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="634 1000 1528 1032">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 250 877 315">Images</h2> <p data-bbox="659 347 1520 500">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 537 1528 607">You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>.</p> <p data-bbox="634 634 1327 667">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="653 711 919 769">Volumes</h2> <p data-bbox="653 808 1482 878">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers.</p> <p data-bbox="653 889 1430 922">One problem occurs when a container crashes or is stopped.</p> <p data-bbox="653 933 1528 1252">Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <u>volume</u> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="634 1279 1306 1312">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="659 269 905 318">Portability</p> <p data-bbox="659 367 1877 488">The flexibility of containers is based on its portability, ease of deployment, and smaller size compared to virtual machines. The Open Container Initiative (OCI), was formed to support fully interoperable container open standards with 3 specifications:</p> <ul data-bbox="676 529 1365 667" style="list-style-type: none"> • The Runtime Specification (runtime-spec) • The Image Specification (image-spec) • The Distribution Specification (distribution-spec) <p data-bbox="659 708 1856 959">The OCI image specification defines an OCI Image, consisting of an image manifest, which contains metadata about contents and dependencies of the image; an image index (optional); a set of filesystem layers; and a configuration, such as arguments and environment variables. You can run the OCI compliant container image on any supported version of Linux or Windows, if you have the OCI compliant container runtime installed on the host.</p> <p data-bbox="634 992 1793 1024">https://docs.aws.amazon.com/whitepapers/latest/containers-on-aws/container-benefits.html</p> <p data-bbox="659 1065 1913 1243">The open container initiative (OCI) images generated by docker build tools will continue to run in your Amazon EKS clusters as before. As an end-user of Kubernetes, you will not experience significant changes. You can continue to use Docker to build your containers outside the cluster. Visit this link to learn why Amazon EKS is discontinuing Dockershim support. For more information, see Kubernetes is Moving on From Dockershim: Commitments and Next Steps on the <i>Kubernetes Blog</i>.</p> <p data-bbox="634 1260 1835 1292">https://aws.amazon.com/blogs/containers/amazon-eks-now-supports-kubernetes-version-1-24/</p>

Claim 1	Accused Instrumentalities
	<p>A container runtime, also known as container engine, is a software component that can run containers on a host operating system. Container runtimes are responsible for loading container images from a repository, monitoring local system resources, isolating system resources for use of a container, and managing container lifecycle. They come in two forms:</p> <ul style="list-style-type: none"> • High-level container runtimes (such as <i>containerd</i> and <i>CRI-O</i>) provide functions that run on top of low-level runtime. • Low-level runtimes are responsible for creating and running containers. The primary job of the low-level container runtimes is to provide container lifecycle management. These runtimes implement the Runtime Specification provided by the OCI(Open Container Initiative), a Linux Foundation project started by Docker, which aims to provide open standards for Linux containers. The default reference implementation for low level runtimes specified by OCI is <i>runc</i>. <p>https://docs.aws.amazon.com/whitepapers/latest/containers-on-aws/key-considerations.html</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="667 256 1297 316"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="667 378 1184 423"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="667 472 1892 545"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="667 581 1900 654"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="632 683 1478 751"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<p data-bbox="648 250 831 289">Overview</p> <p data-bbox="648 345 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="667 628 1908 1008"> <pre data-bbox="667 764 982 857"> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p data-bbox="1077 634 1312 1003">  <div data-bbox="1094 634 1312 959"> /bin/java /opt/app.jar /lib/libc </div> <p data-bbox="1171 979 1234 1008">layer</p> </p></div> <p data-bbox="1339 802 1367 821">+</p> <div data-bbox="1367 634 1619 959">  <pre data-bbox="1388 732 1535 862"> { "manifests": { "platform": { "os": "linux", ... } } } </pre> <p data-bbox="1430 979 1577 1008">image index</p> </div> <p data-bbox="1629 802 1656 821">+</p> <div data-bbox="1661 634 1908 959">  <pre data-bbox="1682 683 1829 911"> { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } } </pre> <p data-bbox="1759 979 1829 1008">config</p> </div>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 245 1297 305">OCI Image Configuration</h2> <p data-bbox="653 358 1913 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="653 558 1661 591">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="632 623 1503 695">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="661 251 745 289">Layer</p> <ul data-bbox="688 326 1915 673" style="list-style-type: none"> • Image filesystems are composed of <i>layers</i>. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="661 722 856 760">Image JSON</p> <ul data-bbox="688 797 1915 1144" style="list-style-type: none"> • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="634 1174 1501 1242">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> • rootfs object, REQUIRED <p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p> <ul style="list-style-type: none"> ◦ type string, REQUIRED <p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p> <ul style="list-style-type: none"> ◦ diff_ids array of strings, REQUIRED <p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>
[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,	<p>In the method practiced by Amazon through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.</p> <p>The system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface. Likewise, Windows applications are compiled to work with Windows, which has a stable binary interface.</p> <p>See discussion and evidence in element [1a] above.</p> <p>See also, e.g.:</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="646 253 1528 298">What are the types of container technology?</p> <p data-bbox="646 329 1593 354">The following are some examples of popular technologies that developers use for containerization.</p> <p data-bbox="646 391 730 415">Docker</p> <p data-bbox="646 453 1913 544">Docker, or Docker Engine, is a popular open-source container runtime that allows software developers to build, deploy, and test containerized applications on various platforms. Docker containers are self-contained packages of applications and related files that are created with the Docker framework.</p> <p data-bbox="646 581 714 605">Linux</p> <p data-bbox="646 643 1904 769">Linux is an open-source operating system with built-in container technology. Linux containers are self-contained environments that allow multiple Linux-based applications to run on a single host machine. Software developers use Linux containers to deploy applications that write or read large amounts of data. Linux containers do not copy the entire operating system to their virtualized environment. Instead, the containers consist of necessary functionalities allocated in the Linux namespace.</p> <p data-bbox="646 807 781 831">Kubernetes</p> <p data-bbox="646 868 1883 959">Kubernetes is a popular open-source container orchestrator that software developers use to deploy, scale, and manage a vast number of microservices. It has a declarative model that makes automating containers easier. The declarative model ensures that Kubernetes takes the appropriate action to fulfil the requirements based on the configuration files.</p> <p data-bbox="632 987 1268 1019">https://aws.amazon.com/what-is/containerization/</p>

Claim 1	Accused Instrumentalities
	<p>How Docker containers work</p> <p>A Docker container is a runtime environment with all the necessary components—like code, dependencies, and libraries—needed to run the application code without using host machine dependencies. This container runtime runs on the engine on a server, machine, or cloud instance. The engine runs multiple containers depending on the underlying resources available.</p> <p>To deploy and scale a set of containers to communicate effectively across different machines or virtual machines, you need a container orchestration platform like Kubernetes. This helps whether your machines are on premises or in the cloud. Kubernetes manages multiple machines, known as a cluster, within the context of container operations.</p> <p>Read about Kubernetes »</p> <p>How Docker images work</p> <p>A Docker image, or container image, is a standalone, executable file used to create a container. This container image contains all the libraries, dependencies, and files that the container needs to run. A Docker image is shareable and portable, so you can deploy the same image in multiple locations at once—much like a software binary file.</p> <p>You can store images in registries to keep track of complex software architectures, projects, business segments, and user group access. For instance, the public Docker Hub registry contains images such as operating systems, programming language frameworks, databases, and code editors.</p> <p>https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/</p> <p>Q: What container images can I run in containers on Bottlerocket?</p> <p>Bottlerocket can run all container images that meet the OCI Image Format specification and Docker images.</p> <p>Q: Can I move my containers running on Amazon Linux 2 to Bottlerocket?</p> <p>Yes, you can move your containers across Amazon Linux 2 and Bottlerocket without modifications.</p> <p>https://aws.amazon.com/bottlerocket/faqs/</p>

Claim 1	Accused Instrumentalities				
	<p>Container technology uses the resource-isolation features of the Linux kernel to sandbox an application, its dependencies, configuration files, and interfaces inside an atomic unit called a container. This allows a container to run on any host with the suitable kernel components, while shielding the application from behavioral inconsistencies through variances in software installed on the host. Containers use operating system (OS) level virtualization compared to VMs, which use hardware level virtualization using hypervisor. A hypervisor is a software or a firmware that creates and runs VMs. Multiple containers can run on a single host OS without needing a hypervisor, while isolated from neighboring containers. This layer of isolation allows consistency, flexibility, and portability, which enable rapid software deployment and testing. There are many ways in which using containers on AWS can benefit your organization. Containers have been widely employed in use cases such as distributed applications, batch jobs, and continuous deployment pipelines. The use cases for containers continue to grow in areas like distributed data processing, streaming media delivery, genomics, and machine learning, including generative AI.</p> <p>https://docs.aws.amazon.com/whitepapers/latest/containers-on-aws/containers-on-aws.html</p>				
<p>[1c] the containers of application software excluding a kernel,</p>	<p>In the method practiced by Amazon and/or its customer through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See discussion and evidence in element [1a] above.</i></p> <p><i>See also, e.g.:</i></p> <div data-bbox="636 1013 1923 1305"> <p>Summary of difference: Docker vs. VM</p> <table> <tr> <th data-bbox="636 1110 1461 1175">Docker container</th><th data-bbox="1461 1110 1923 1175">VM</th></tr> <tr> <td data-bbox="636 1175 1461 1305">Architecture Shares resources with the underlying host kernel.</td><td data-bbox="1461 1175 1923 1305">Runs its own kernel and operating system.</td></tr> </table> </div> <p>https://aws.amazon.com/compare/the-difference-between-docker-vm/</p>	Docker container	VM	Architecture Shares resources with the underlying host kernel.	Runs its own kernel and operating system.
Docker container	VM				
Architecture Shares resources with the underlying host kernel.	Runs its own kernel and operating system.				

Claim 1	Accused Instrumentalities
	<p>Docker containers, on the other hand, use resources on demand. Rather than asking for a specific amount of physical hardware resourcing as virtual machines do, they simply request what they need from the single operating system kernel. Multiple containers share the same operating system. Docker containers direct resource sharing with the kernel leads and may use less system resources compared to a VM.</p> <p>Security</p> <p>Because Docker containers share the kernel with the host operating system, for lightweight resource consumption, they're at risk if there are vulnerabilities in the kernel. However, Docker also provides many advanced security controls.</p> <p>https://aws.amazon.com/compare/the-difference-between-docker-vm/</p>

Claim 1	Accused Instrumentalities																																			
	<table><tr><th>Criteria</th><th>EKS managed node groups</th><th>Self managed nodes</th></tr><tr><td>Can be deployed to AWS Outposts</td><td>No</td><td>Yes</td></tr><tr><td>Can be deployed to an AWS Local Zone</td><td>No</td><td>Yes – For more information, see Amazon EKS and AWS Local Zones.</td></tr><tr><td>Can run containers that require Windows</td><td>Yes</td><td>Yes – Your cluster still requires at least one (two recommended for availability) Linux node though.</td></tr><tr><td>Can run containers that require Linux</td><td>Yes</td><td>Yes</td></tr><tr><td>Can run workloads that require the Inferentia chip</td><td>Yes – Amazon Linux nodes only</td><td>Yes – Amazon Linux only</td></tr><tr><td>Can run workloads that require a GPU</td><td>Yes – Amazon Linux nodes only</td><td>Yes – Amazon Linux only</td></tr><tr><td>Can run workloads that require Arm processors</td><td>Yes</td><td>Yes</td></tr><tr><td>Can run AWS Bottlerocket</td><td>Yes</td><td>Yes</td></tr><tr><td>Pods share a kernel runtime environment with other Pods</td><td>Yes – All of your Pods on each of your nodes</td><td>Yes – All of your Pods on each of your nodes</td></tr><tr><td>Pods share CPU, memory, storage, and network resources with other Pods.</td><td>Yes – Can result in unused resources on each node</td><td>Yes – Can result in unused resources on each node</td></tr></table>	Criteria	EKS managed node groups	Self managed nodes	Can be deployed to AWS Outposts	No	Yes	Can be deployed to an AWS Local Zone	No	Yes – For more information, see Amazon EKS and AWS Local Zones .	Can run containers that require Windows	Yes	Yes – Your cluster still requires at least one (two recommended for availability) Linux node though.	Can run containers that require Linux	Yes	Yes	Can run workloads that require the Inferentia chip	Yes – Amazon Linux nodes only	Yes – Amazon Linux only	Can run workloads that require a GPU	Yes – Amazon Linux nodes only	Yes – Amazon Linux only	Can run workloads that require Arm processors	Yes	Yes	Can run AWS Bottlerocket	Yes	Yes	Pods share a kernel runtime environment with other Pods	Yes – All of your Pods on each of your nodes	Yes – All of your Pods on each of your nodes	Pods share CPU, memory, storage, and network resources with other Pods.	Yes – Can result in unused resources on each node	Yes – Can result in unused resources on each node	https://docs.aws.amazon.com/eks/latest/userguide/eks-compute.html	
Criteria	EKS managed node groups	Self managed nodes																																		
Can be deployed to AWS Outposts	No	Yes																																		
Can be deployed to an AWS Local Zone	No	Yes – For more information, see Amazon EKS and AWS Local Zones .																																		
Can run containers that require Windows	Yes	Yes – Your cluster still requires at least one (two recommended for availability) Linux node though.																																		
Can run containers that require Linux	Yes	Yes																																		
Can run workloads that require the Inferentia chip	Yes – Amazon Linux nodes only	Yes – Amazon Linux only																																		
Can run workloads that require a GPU	Yes – Amazon Linux nodes only	Yes – Amazon Linux only																																		
Can run workloads that require Arm processors	Yes	Yes																																		
Can run AWS Bottlerocket	Yes	Yes																																		
Pods share a kernel runtime environment with other Pods	Yes – All of your Pods on each of your nodes	Yes – All of your Pods on each of your nodes																																		
Pods share CPU, memory, storage, and network resources with other Pods.	Yes – Can result in unused resources on each node	Yes – Can result in unused resources on each node																																		

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> • Container runtime – Software running on a container host (virtual machine or bare metal server) operating system that is responsible for running and managing containers. The container relies on the kernel of the host for all system calls. https://docs.aws.amazon.com/wellarchitected/latest/container-build-lens/container-technology-terminology.html <p>6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p>Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers. https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p>
<p>[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Amazon and/or its customer through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.</p> <p>For example, each container will utilize its own local system files, including libraries such as libc/glibc (or Windows equivalents, such as DLLs) and configuration files, not the corresponding libraries and configuration files of the host OS.</p> <p><i>See discussion and evidence in element [1a] above.</i></p> <p><i>See also, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="674 248 1780 310">Architect your application for Amazon ECS</h2> <p data-bbox="674 334 793 358"> PDF RSS </p> <p data-bbox="674 427 1860 493"> You architect your application by creating a task definition for your application. The task definition contains the parameters that define information about the application, including: </p> <ul data-bbox="695 532 1887 688" style="list-style-type: none"> • The launch type to use, which determines the infrastructure that your tasks are hosted on. When you use the EC2 launch type, you also choose the instance type. For some instance types, such as GPU, you need to set additional parameters. For more information, see Amazon ECS task definition use cases. • The container image, which holds your application code and all the dependencies that your application code requires to run. • The networking mode to use for the containers in your task The networking mode determines how your task communicates over the network. For tasks that run on EC2 instance, there are multiple options, but we recommend that you use the <code>awsvpc</code> network mode. The <code>awsvpc</code> network mode simplifies container networking, because you have more control over how your applications communicate with each other and other services within your VPCs. For tasks that run on Fargate, you can only use the <code>awsvpc</code> network mode. <ul data-bbox="695 1110 1482 1187" style="list-style-type: none"> • The logging configuration to use for your tasks. • Any data volumes that are used with the containers in the task. <p data-bbox="634 1206 1848 1235"> https://docs.aws.amazon.com/AmazonECS/latest/developerguide/application_architecture.html </p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="667 256 1213 316">Container image type</h2> <p data-bbox="667 337 783 365"> PDF RSS </p> <p data-bbox="667 427 1843 573"> The time that it takes a container to start up varies, based on the underlying container image. For example, a fatter image (full versions of Debian, Ubuntu, and Amazon1/2) might take longer to start up because there are a more services that run in the containers compared to their respective slim versions (Debian-slim, Ubuntu-slim, and Amazon-slim) or smaller base images (Alpine). </p> <p data-bbox="634 625 1764 657"> https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/container-type.html </p> <ul data-bbox="655 695 1911 1320" style="list-style-type: none"> • Container – Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. Containers share an operating system installed on the host server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of environment. A container is a runnable instance of an image (see Deep Dive on Containers on AWS getting started resource center). While a container image is immutable, the container adds a read/write layer on top of the image to which your application can write information like temporary files or logs (see Docker overview). When the container stops and is removed, the information written to the temporary read/write layer will be lost. • Container image – Container images are read-only templates used to build out containers. Container images are immutable, meaning they cannot be changed once created. Container images are created using layers by reading a text file that is called a Dockerfile that contains all necessary information. You can find the Dockerfile reference here. • Container runtime – Software running on a container host (virtual machine or bare metal server) operating system that is responsible for running and managing containers. The container relies on the kernel of the host for all system calls. • Dockerfile – A file or series of files containing commands that describe the content of a container image. Each command represents a layer on the Container image (see the <i>Container image layers</i> definition). • Container build – A container image built from a Dockerfile. This process results in a container image containing the necessary components to run your containerized application. • Base image – A starting point container image that is used in the container build process to generate custom or new container images. This image has FROM scratch in the Dockerfile. <p data-bbox="634 1328 1843 1401"> https://docs.aws.amazon.com/wellarchitected/latest/container-build-lens/container-technology-terminology.html </p>

Claim 1	Accused Instrumentalities
	<p>Base image</p> <p>The base image is the selected image and operating system used in your image or container recipe document, along with the components. The base image and the component definitions combined produce the desired configuration for the output image.</p> <p>https://docs.aws.amazon.com/imagebuilder/latest/userguide/what-is-image-builder.html</p> <p>How does containerization work?</p> <p>Containerization involves building self-sufficient software packages that perform consistently, regardless of the machines they run on. Software developers create and deploy container images—that is, files that contain the necessary information to run a containerized application. Developers use containerization tools to build container images based on the Open Container Initiative (OCI) image specification. OCI is an open-source group that provides a standardized format for creating container images. Container images are read-only and cannot be altered by the computer system.</p> <p>Container images are the top layer in a containerized system that consists of the following layers.</p> <p>https://aws.amazon.com/what-is/containerization/</p>

Claim 1	Accused Instrumentalities
	<p>Infrastructure</p> <p>Infrastructure is the hardware layer of the container model. It refers to the physical computer or bare-metal server that runs the containerized application.</p> <p>Operating system</p> <p>The second layer of the containerization architecture is the operating system. Linux is a popular operating system for containerization with on-premise computers. In cloud computing, developers use cloud services such as AWS EC2 to run containerized applications.</p> <p>Container engine</p> <p>The container engine, or container runtime, is a software program that creates containers based on the container images. It acts as an intermediary agent between the containers and the operating system, providing and managing resources that the application needs. For example, container engines can manage multiple containers on the same operating system by keeping them independent of the underlying infrastructure and each other.</p> <p>Application and dependencies</p> <p>The topmost layer of the containerization architecture is the application code and the other files it needs to run, such as library dependencies and related configuration files. This layer might also contain a light guest operating system that gets installed over the host operating system.</p> <p>https://aws.amazon.com/what-is/containerization/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="646 253 1528 302">What are the types of container technology?</p> <p data-bbox="646 329 1593 354">The following are some examples of popular technologies that developers use for containerization.</p> <p data-bbox="646 391 730 415">Docker</p> <p data-bbox="646 453 1913 545">Docker, or Docker Engine, is a popular open-source container runtime that allows software developers to build, deploy, and test containerized applications on various platforms. Docker containers are self-contained packages of applications and related files that are created with the Docker framework.</p> <p data-bbox="646 583 714 607">Linux</p> <p data-bbox="646 644 1904 769">Linux is an open-source operating system with built-in container technology. Linux containers are self-contained environments that allow multiple Linux-based applications to run on a single host machine. Software developers use Linux containers to deploy applications that write or read large amounts of data. Linux containers do not copy the entire operating system to their virtualized environment. Instead, the containers consist of necessary functionalities allocated in the Linux namespace.</p> <p data-bbox="646 807 781 831">Kubernetes</p> <p data-bbox="646 868 1883 961">Kubernetes is a popular open-source container orchestrator that software developers use to deploy, scale, and manage a vast number of microservices. It has a declarative model that makes automating containers easier. The declarative model ensures that Kubernetes takes the appropriate action to fulfil the requirements based on the configuration files.</p> <p data-bbox="632 989 1268 1021">https://aws.amazon.com/what-is/containerization/</p>

Claim 1	Accused Instrumentalities
	<p>How Docker containers work</p> <p>A Docker container is a runtime environment with all the necessary components—like code, dependencies, and libraries—needed to run the application code without using host machine dependencies. This container runtime runs on the engine on a server, machine, or cloud instance. The engine runs multiple containers depending on the underlying resources available.</p> <p>To deploy and scale a set of containers to communicate effectively across different machines or virtual machines, you need a container orchestration platform like Kubernetes. This helps whether your machines are on premises or in the cloud. Kubernetes manages multiple machines, known as a cluster, within the context of container operations.</p> <p>Read about Kubernetes »</p> <p>How Docker images work</p> <p>A Docker image, or container image, is a standalone, executable file used to create a container. This container image contains all the libraries, dependencies, and files that the container needs to run. A Docker image is shareable and portable, so you can deploy the same image in multiple locations at once—much like a software binary file.</p> <p>You can store images in registries to keep track of complex software architectures, projects, business segments, and user group access. For instance, the public Docker Hub registry contains images such as operating systems, programming language frameworks, databases, and code editors.</p> <p>https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/</p>
<p>[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Amazon and/or its customer through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p>For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same version of libc (<i>e.g.</i>, where both host and container use the same version of Amazon Linux). In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container.</p>

Claim 1	Accused Instrumentalities
	<p><i>See</i> discussion and evidence in element [1a] above.</p> <p><i>See also, e.g.:</i></p> <p>How Docker containers work</p> <p>A Docker container is a runtime environment with all the necessary components—like code, dependencies, and libraries—needed to run the application code without using host machine dependencies. This container runtime runs on the engine on a server, machine, or cloud instance. The engine runs multiple containers depending on the underlying resources available.</p> <p>To deploy and scale a set of containers to communicate effectively across different machines or virtual machines, you need a container orchestration platform like Kubernetes. This helps whether your machines are on premises or in the cloud. Kubernetes manages multiple machines, known as a cluster, within the context of container operations.</p> <p>Read about Kubernetes »</p> <p>How Docker images work</p> <p>A Docker image, or container image, is a standalone, executable file used to create a container. This container image contains all the libraries, dependencies, and files that the container needs to run. A Docker image is shareable and portable, so you can deploy the same image in multiple locations at once—much like a software binary file.</p> <p>You can store images in registries to keep track of complex software architectures, projects, business segments, and user group access. For instance, the public Docker Hub registry contains images such as operating systems, programming language frameworks, databases, and code editors.</p> <p>https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="667 248 1577 310">Amazon ECS-optimized Linux AMIs</h2> <p data-bbox="667 334 789 362"> PDF RSS </p> <p data-bbox="667 423 1902 773"> Amazon ECS provides the Amazon ECS-optimized AMIs that are preconfigured with the requirements and recommendations to run your container workloads. We recommend that you use the Amazon ECS-optimized Amazon Linux 2023 AMI for your Amazon EC2 instances unless your application requires Amazon EC2 GPU-based instances, a specific operating system or a Docker version that is not yet available in that AMI. For information about the Amazon Linux 2 and Amazon Linux 2023 instances, see Comparing Amazon Linux 2 and Amazon Linux 2023 in the <i>Amazon Linux 2023 User Guide</i>. Launching your container instances from the most recent Amazon ECS-Optimized AMI ensures that you receive the current security updates and container agent version. For information about how to launch an instance, see Launching an Amazon ECS Linux container instance. </p> <p data-bbox="634 800 1797 833"> https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-optimized_AMI.html </p> <h2 data-bbox="695 889 1709 951">Using the AL2023 base container image</h2> <p data-bbox="695 976 816 1003"> PDF RSS </p> <p data-bbox="695 1060 1892 1206"> The AL2023 container image is built from the same software components that are included in the AL2023 AMI. It's available for use in any environment as a base image for Docker workloads. If you're using the Amazon Linux AMI for applications in Amazon Elastic Compute Cloud (Amazon EC2), you can containerize your applications with the Amazon Linux container image. </p> <p data-bbox="634 1230 1480 1263"> https://docs.aws.amazon.com/linux/al2023/ug/base-container.html </p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="688 245 1751 305">Pulling the Amazon Linux container image</h2> <p data-bbox="688 329 806 354">PDF RSS</p> <p data-bbox="688 415 1877 558">The Amazon Linux container image is built from the same software components that are included in the Amazon Linux AMI. The Amazon Linux container image is available for use in any environment as a base image for Docker workloads. If you use the Amazon Linux AMI for applications in Amazon EC2, you can containerize your applications with the Amazon Linux container image.</p> <p data-bbox="688 594 1877 698">You can use the Amazon Linux container image in your local development environment and then push your application to AWS using Amazon ECS. For more information, see Using Amazon ECR images with Amazon ECS.</p> <p data-bbox="634 717 1877 748">https://docs.aws.amazon.com/AmazonECR/latest/userguide/amazon_linux_container_image.html</p> <p data-bbox="634 779 1486 898"><code>COPY</code> and <code>ADD</code>: These commands copy files and directories from your local filesystem into the Docker image. They are often used to include your application code, configuration files, and dependencies.</p> <p data-bbox="634 909 1919 976">https://medium.com/@swalperen3008/what-is-dockerize-and-dockerize-your-project-a-step-by-step-guide-899c48a34df6</p> <h2 data-bbox="667 1013 978 1057">Container images</h2> <p data-bbox="667 1088 1264 1214">A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="634 1239 1230 1269">https://kubernetes.io/docs/concepts/containers/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 248 1272 316">About storage drivers</h2> <p data-bbox="646 362 1871 488">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 557 1564 613">Storage drivers versus Docker volumes</h2> <p data-bbox="646 651 1913 919">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 967 1902 1094">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1016 1535 1040" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1122 1224 1154"><a data-bbox="636 1122 1224 1154" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 245 1081 302">Images and layers</h2> <p data-bbox="657 337 1822 415">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 483 1451 797"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 862 1900 1170">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="632 1192 1224 1224">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p> <div data-bbox="892 724 1635 1300" data-label="Diagram"> <p>The diagram illustrates the Docker layer architecture. It shows a stack of four blue rectangular blocks representing image layers, each with a unique hash and a size. From bottom to top, the layers are: <code>d3a1f33e8a5a</code> (188.1 MB), <code>c22013c84729</code> (194.5 KB), <code>d74508fb6632</code> (1.895 KB), and <code>91e54dfb1179</code> (0 B). To the right of this stack is a bracket labeled "Image Layers (R/O)" with a padlock icon, indicating they are read-only. Above the stack is a dashed rectangular box labeled "Thin R/W layer". A bracket to the right of this box is labeled "Container layer". Five double-headed vertical arrows connect the top of the "Thin R/W layer" to the top of each of the four image layers below it. Below the entire stack is the text "ubuntu:15.04" and "Container (based on ubuntu:15.04 image)".</p> </div> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
<p>[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,</p>	<p>In the method practiced by Amazon through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p>For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software.</p> <p><i>See, e.g.:</i></p> <p>Container technology uses the resource-isolation features of the Linux kernel to sandbox an application, its dependencies, configuration files, and interfaces inside an atomic unit called a container. This allows a container to run on any host with the suitable kernel components, while shielding the application from behavioral inconsistencies through variances in software installed on the host. Containers use operating system (OS) level virtualization compared to VMs, which use hardware level virtualization using hypervisor. A hypervisor is a software or a firmware that creates and runs VMs. Multiple containers can run on a single host OS without needing a hypervisor, while isolated from neighboring containers. This layer of isolation allows consistency, flexibility, and portability, which enable rapid software deployment and testing. There are many ways in which using containers on AWS can benefit your organization. Containers have been widely employed in use cases such as distributed applications, batch jobs, and continuous deployment pipelines. The use cases for containers continue to grow in areas like distributed data processing, streaming media delivery, genomics, and machine learning, including generative AI.</p> <p>https://docs.aws.amazon.com/whitepapers/latest/containers-on-aws/containers-on-aws.html</p>

Claim 1	Accused Instrumentalities
	<p>Before we get too deep into technical details, I want to talk about how containers are typically used and why we see some consistent feedback about those themes. In any environment, booting a computer can take a while. But what's harder than booting is deploying a random application to that computer, and doing so reliably. Containers make this process a lot easier. A container image provides a reliable and repeatable mechanism for packaging up the set of local dependencies for an application, including its dynamically linked libraries, other programs to invoke, and assets. The Linux kernel primitives that power containers, including cgroups and namespaces, provide some amount of resource and visibility isolation. Containers also start up much more quickly than a whole computer. These properties enable each application to pretend that it's the only application running, enables subdividing larger computers into smaller parts so more of these applications can run together without conflict, and makes it attractive to use one computer for running multiple applications or even a cluster of computers to run many copies of those applications.</p> <p>https://aws.amazon.com/blogs/containers/bottlerocket-a-special-purpose-container-operating-system/</p> <h2 data-bbox="667 721 1325 773">Build secure microservices</h2> <p>Ensure strong security isolation between your containers. AWS provides the latest security updates and lets you set granular access permissions for every container. AWS offers over 210 security, compliance, and governance services, plus key features to best suit your needs.</p> <p>https://aws.amazon.com/containers/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="653 245 871 282">Understand</p> <p data-bbox="653 326 1673 354">Containers offer a number of advantages for packaging, deploying, and running applications:</p> <ul data-bbox="653 386 1885 451" style="list-style-type: none"> • Isolation: Improve security and reliability with containers' process-level isolation, with which applications running in separate containers cannot interfere with each other, improving security and reliability. <p data-bbox="632 467 1871 537">https://docs.aws.amazon.com/decision-guides/latest/containers-on-aws-how-to-choose/choosing-aws-container-service.html</p> <p data-bbox="653 578 825 605">Fault tolerance</p> <p data-bbox="653 643 1892 732">Software development teams use containers to build fault-tolerant applications. They use multiple containers to run microservices on the cloud. Because containerized microservices operate in isolated user spaces, a single faulty container doesn't affect the other containers. This increases the resilience and availability of the application.</p> <p data-bbox="653 768 730 795">Agility</p> <p data-bbox="653 833 1892 922">Containerized applications run in isolated computing environments. Software developers can troubleshoot and change the application code without interfering with the operating system, hardware, or other application services. They can shorten software release cycles and work on updates quickly with the container model.</p> <p data-bbox="632 959 1268 992">https://aws.amazon.com/what-is/containerization/</p> <p data-bbox="653 1036 1913 1105">Linux containers are made up of control groups (cgroups) and namespaces that help limit what a container can access, but all containers share the same Linux kernel as the host Amazon EC2 instance.</p> <p data-bbox="632 1122 1444 1154">https://docs.aws.amazon.com/eks/latest/userguide/security.html</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="653 240 1772 321">Use Windows containers instead of running many applications on one instance of IIS</p> <p data-bbox="653 354 1866 418">Consider the following advantages of using Windows containers instead of running multiple applications on one EC2 Windows instance with Internet Information Services (IIS):</p> <ul data-bbox="653 451 1898 979" style="list-style-type: none"> <li data-bbox="653 451 1898 597">• Security – Containers provide a level of security out of the box that isn't achieved through isolation at the IIS level. If one IIS website or application is compromised, all the other hosted sites are exposed and vulnerable. Container escape is rare and a harder vulnerability to exploit than gaining control of a server through a web vulnerability. <li data-bbox="653 605 1898 719">• Flexibility – The ability to run containers in process isolation and have their own instance allows for more granular networking options. Containers also offer complex distribution methods across many EC2 instances. You don't get these benefits when you consolidate applications on a single IIS instance. <li data-bbox="653 727 1898 979">• Management overhead – Server Name Indication (SNI) creates overhead that requires management and automation. Also, you have to grapple with typical operating system management operations like patching, troubleshooting BSOD (if auto scaling isn't in place), endpoint protection, and so on. Configuring IIS sites according to security best practices is a time consuming and ongoing activity. You might even need to set up trust levels, which also adds to management overhead. Containers are designed to be stateless and immutable. Ultimately, your deployments are faster, more secure, and repeatable if you use Windows containers instead. <p data-bbox="632 987 1709 1060">https://docs.aws.amazon.com/prescriptive-guidance/latest/optimize-costs-microsoft-workloads/windows-containers-main.html</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 248 1272 316">About storage drivers</h2> <p data-bbox="646 363 1871 488">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 557 1564 613">Storage drivers versus Docker volumes</h2> <p data-bbox="646 651 1913 919">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 967 1902 1092">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1015 1535 1040" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1122 1224 1154"><a data-bbox="636 1122 1224 1154" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 245 1081 302">Images and layers</h2> <p data-bbox="657 337 1822 415">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 483 1453 797"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 862 1900 1170">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1192 1226 1224">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p> <div data-bbox="892 724 1635 1300" data-label="Diagram"> <p>The diagram illustrates the Docker layer architecture. It shows a stack of four blue rectangular blocks representing image layers, each with a unique ID and a size. From bottom to top, the layers are: <code>d3a1f33e8a5a</code> (188.1 MB), <code>c22013c84729</code> (194.5 KB), <code>d74508fb6632</code> (1.895 KB), and <code>91e54dfb1179</code> (0 B). To the right of this stack is a bracket labeled "Image Layers (R/O)" with a padlock icon, indicating they are read-only. Above the stack is a dashed box labeled "Thin R/W layer" with an arrow pointing to it from the label "Container layer". Five double-headed vertical arrows connect the top of the "Thin R/W layer" to the top of each of the four image layers below it. Below the entire stack is the text "ubuntu:15.04" and "Container (based on ubuntu:15.04 image)".</p> </div> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by Amazon through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p>For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system.</p> <p><i>See, e.g.:</i></p> <p>Amazon EC2 instance root volume</p> <p>PDF RSS</p> <p>When you launch an instance, we create a <i>root volume</i> for the instance. The root volume contains the image used to boot the instance. Each instance has a single root volume. You can add storage volumes to your instances during or after launch.</p> <p>https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/RootDeviceStorage.html</p>

Claim 1

Accused Instrumentalities

Storage options for Amazon ECS tasks

PDF | RSS

Amazon ECS provides you with flexible, cost effective, and easy-to-use data storage options depending on your needs. Amazon ECS supports the following data volume options for containers:

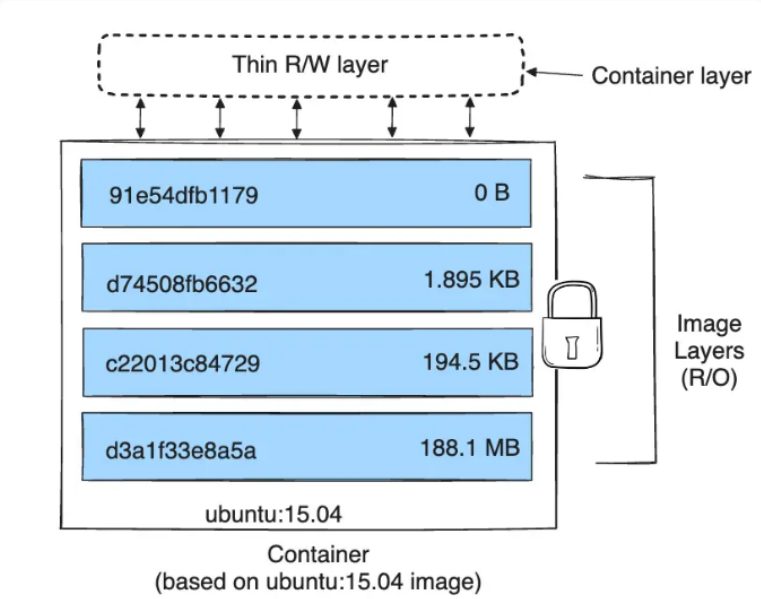
Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
Amazon Elastic Block Store (Amazon EBS)	Fargate, Amazon EC2	Linux	Can be persisted when attached to a standalone task. Ephemeral when attached to a task maintained by a service.	Amazon EBS volumes provide cost-effective, durable, high-performance block storage for data-intensive containerized workloads. Common use cases include transactional workloads such as databases, virtual desktops and root volumes, and throughput intensive workloads such as log processing and ETL workloads. For more information, see Use Amazon EBS volumes with Amazon ECS .

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/using_data_volumes.html

Claim 1	Accused Instrumentalities
	<p>To use a volume, specify the volumes to provide for the Pod in <code>.spec.volumes</code> and declare where to mount those volumes into containers in <code>.spec.containers[*].volumeMounts</code>. A process in a container sees a filesystem view composed from the initial contents of the container image, plus volumes (if defined) mounted inside the container. The process sees a root filesystem that initially matches the contents of the container image. Any writes to within that filesystem hierarchy, if allowed, affect what that process views when it performs a subsequent filesystem access. Volumes mount at the specified paths within the image. For each container defined within a Pod, you must independently specify where to mount each volume that the container uses.</p> <p>https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="642 248 1272 318">About storage drivers</h2> <p data-bbox="642 363 1871 488">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="642 557 1566 610">Storage drivers versus Docker volumes</h2> <p data-bbox="642 651 1913 919">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="642 967 1902 1092">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="642 1122 1224 1154">https://docs.docker.com/storage/storagedriver/</p>




Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 245 1081 302">Images and layers</h2> <p data-bbox="657 337 1822 415">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 483 1451 797"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 862 1900 1170">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="632 1192 1226 1224">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a Docker container. At the top is a dashed box labeled "Thin R/W layer" (Container layer). Below it is a stack of four solid blue boxes representing image layers, each with a hash and a size: 91e54dfb1179 (0 B), d74508fb6632 (1.895 KB), c22013c84729 (194.5 KB), and d3a1f33e8a5a (188.1 MB). The bottom layer is labeled "ubuntu:15.04". A padlock icon is shown next to the stack, labeled "Image Layers (R/O)". The entire stack is labeled "Container (based on ubuntu:15.04 image)".</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>The original purpose of the cgroup, chroot, and namespace facilities in the kernel was to protect applications from noisy, nosey, and messy neighbors. Combining these with container images created an abstraction that also isolates applications from the [heterogeneous] operating systems</p> <p>https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2>Container environment</h2> <p>The Kubernetes Container environment provides several important resources to Containers:</p> <ul style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p>https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 250 877 315">Images</h2> <p data-bbox="659 347 1522 500">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 535 1528 607">You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>.</p> <p data-bbox="634 636 1329 667">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="653 711 919 769">Volumes</h2> <p data-bbox="653 808 1482 880">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers.</p> <p data-bbox="653 889 1430 920">One problem occurs when a container crashes or is stopped.</p> <p data-bbox="653 930 1528 1252">Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <u>Pod</u> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <u>volume</u> abstraction solves both of these problems. Familiarity with <u>Pods</u> is suggested.</p> <p data-bbox="634 1281 1308 1312">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="674 256 1297 315">Open Container Initiative</h2> <hr data-bbox="674 329 1908 332"/> <h3 data-bbox="674 380 1182 422">Image Format Specification</h3> <hr data-bbox="674 435 1908 438"/> <p data-bbox="674 472 1908 545">This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p> <p data-bbox="674 581 1908 654">The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p> <p data-bbox="634 683 1480 756">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="648 250 831 289">Overview</p> <p data-bbox="648 345 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="667 626 1908 1008"> <pre data-bbox="667 764 982 857"> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p data-bbox="1077 634 1310 959">  /bin/java /opt/app.jar /lib/libc </p> <p data-bbox="1171 979 1234 1008">layer</p> <p data-bbox="1339 805 1367 826">+</p> <p data-bbox="1381 634 1614 959">  { "manifests": { "platform": { "os": "linux", ... } } } </p> <p data-bbox="1430 979 1577 1008">image index</p> <p data-bbox="1629 805 1656 826">+</p> <p data-bbox="1671 634 1908 959">  { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } } </p> <p data-bbox="1759 979 1829 1008">config</p> </div> <p data-bbox="632 1037 1478 1105"> https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md </p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 245 1297 305">OCI Image Configuration</h2> <p data-bbox="653 358 1913 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="653 558 1661 591">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="632 623 1503 695">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="661 251 751 289">Layer</p> <ul data-bbox="688 326 1919 672" style="list-style-type: none"> • Image filesystems are composed of <i>layers</i>. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="661 722 856 760">Image JSON</p> <ul data-bbox="688 797 1919 1143" style="list-style-type: none"> • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="632 1174 1503 1243">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> • rootfs object, REQUIRED <p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p> <ul style="list-style-type: none"> ◦ type string, REQUIRED <p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p> <ul style="list-style-type: none"> ◦ diff_ids array of strings, REQUIRED <p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 2

Claim 2	Accused Instrumentalities
<p>2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.</p>	<p>Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.</p> <p>For example, a container image has an associated image configuration comprising information for starting the one or more applications. This can be an Open Containers Initiative image configuration.</p> <p><i>See, e.g.:</i></p>

Claim 2	Accused Instrumentalities
	<div data-bbox="659 256 1283 313"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="659 378 1169 423"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="659 472 1877 545"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="659 581 1885 654"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="621 683 1465 751"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 2	Accused Instrumentalities
	<p data-bbox="638 250 819 289">Overview</p> <p data-bbox="638 345 1879 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="653 626 1896 1008"> <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p>→</p> <div style="display: flex; align-items: center; justify-content: space-around;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="background-color: #000080; color: white; padding: 2px 5px;">Ci</div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 100px; height: 100px; margin: 10px auto; display: flex; align-items: center; justify-content: center;"> <div style="text-align: left; padding: 5px;"> /bin/java /opt/app.jar /lib/libc </div> </div> <p>layer</p> </div> <div style="font-size: 2em;">+</div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="background-color: #000080; color: white; padding: 2px 5px;">Ci</div> </div> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: 150px;"> { "manifests": { "platform": { "os": "linux", ... } } </div> <p>image index</p> </div> <div style="font-size: 2em;">+</div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="background-color: #000080; color: white; padding: 2px 5px;">Ci</div> </div> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: 150px;"> { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } } </div> <p>config</p> </div> </div> </div> <p data-bbox="621 1036 1465 1105"> https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md </p>

Claim 2	Accused Instrumentalities
	<h2 data-bbox="640 245 1283 305">OCI Image Configuration</h2> <p data-bbox="640 358 1902 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="640 558 1646 591">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="621 623 1493 695">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 2	Accused Instrumentalities
	<ul style="list-style-type: none"> • config object, OPTIONAL <p>The execution parameters which SHOULD be used as a base when running a container using the image. This field can be <code>null</code>, in which case any execution parameters should be specified at creation of the container.</p> <ul style="list-style-type: none"> ◦ Env array of strings, OPTIONAL <p>Entries are in the format of <code>VARNAME=VARVALUE</code>. These values act as defaults and are merged with any specified when creating a container.</p> <ul style="list-style-type: none"> ◦ Entrypoint array of strings, OPTIONAL <p>A list of arguments to use as the command to execute when the container starts. These values act as defaults and may be replaced by an entrypoint specified when creating a container.</p> <ul style="list-style-type: none"> ◦ Cmd array of strings, OPTIONAL <p>Default arguments to the entrypoint of the container. These values act as defaults and may be replaced by any specified when creating a container. If an <code>Entrypoint</code> value is not specified, then the first entry of the <code>Cmd</code> array SHOULD be interpreted as the executable to run.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 4

Claim 4	Accused Instrumentalities
<p>4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.</p>	<p>Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.</p> <p>For example, Amazon's App2Container functionality identifies an application along with its dependencies to be migrated to the target container. This identification step happens before storing the containers having the migrated application and files in the target machine, as described above.</p> <p><i>See analysis and evidence for claim 1 above.</i></p> <p><i>See also, e.g.:</i></p> <p>Q: What is AWS App2Container ?</p> <p>AWS App2Container (A2C) is a new command line tool that helps transform existing applications running in virtual machines into containers, without needing any code changes. A2C discovers applications running on a server, identifies dependencies, and generates relevant artifacts for seamless deployment to Amazon ECS and EKS. A2C also provides integration with AWS CodeBuild and CodeDeploy to enable a repeatable way to build and deploy containerized application.</p> <p>https://aws.amazon.com/app2container/faqs/</p>

Claim 4	Accused Instrumentalities
	<p>Q: How does App2Container make it easier to containerize applications?</p> <p>App2Container enables you to quickly transform your existing applications running on-premises, in Amazon EC2, or in any other cloud. You get the following benefits by using A2C for containerization:</p> <ul style="list-style-type: none"> • Application Inventory: A2C identifies the supported ASP.NET and Java applications running in a server which enables a quick and accurate inventory of applications in your environment without extensive manual effort. • Detailed Dependency Analysis: A2C analyzes the running application and identifies dependencies including cooperating processes and network port dependencies. This detailed information reduces the manual effort required to understand and document application anatomy and required dependencies. • Seamless Deployment: A2C generates ECS task definitions and Kubernetes deployment YAML for the containerized application following the AWS best practices for security and scalability by integrating with various AWS services such as ECR, ECS, and EKS. A2C generates CloudFormation template to configure required compute, network, and security infrastructure to seamlessly deploy containerized application in AWS. A2C creates CI/CD pipelines for Amazon DevOps services such as CodeBuild and CodeDeploy to build and deploy containers. If you have existing CI/CD tooling (for example, Azure DevOps, and Jenkins), then you can integrate A2C provided deployment artifacts into your existing CI/CD workflows. <p>https://aws.amazon.com/app2container/faqs/</p>

Claim 4	Accused Instrumentalities
	<p>Q: How do I get started?</p> <p>Once the prerequisites are met and the App2Container tool is installed on the application server, use the following sequence of commands to containerize the application.</p> <ul style="list-style-type: none"> • Init: Configures the tool with your AWS profile and other related parameters. • Inventory: Identifies all running applications that are supported by A2C. Choose an application to proceed. • Analyze: Performs detailed analysis to identify application artifacts, third-party dependencies, network ports and configuration files, and generates a configuration file to represent the gathered information. You can make additional changes to the configuration file (for example, exclude or include any additional files or directories). • Containerize: Generates a dockerfile and container image for the analyzed application. If needed, you can modify the A2C-generated dockerfile and rebuild the container. You can also perform local testing of the containerized application using docker run command before deploying to AWS container services. • Generate Deployment: Registers the container image with Amazon ECR, creates ECS task definitions, and Kubernetes deployment YAML file to deploy the containerized application to Amazon ECS or Amazon EKS. A2C generates CloudFormation template to create required infrastructure to deploy the containerized application. Optionally, it also generates AWS CodeBuild and CodeDeploy artifacts to enable repeatable way to build and deploy the containerized app. <p>https://aws.amazon.com/app2container/faqs/</p> <p>Q: What artifacts are generated by App2Container?</p> <p>App2Container generates the following artifacts for each application component: 1) Application artifacts such as DLLs configuration, 2) Dockerfile, 3) container image in ECR, 4) ECS Task definitions, 5) Kubernetes deployment YAML, 6) CloudFormation Template, 6) AWS CodeBuild and CodeDeploy resources.</p> <p>https://aws.amazon.com/app2container/faqs/</p>

Claim 6

Claim 6	Accused Instrumentalities
6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of	Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

Claim 6	Accused Instrumentalities
<p>a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.</p>	<p>For example, Kubernetes containers have an associated hostname, which in the case of a single-container Pod is the unique identity of that container. For another example, Kubernetes pods have an associated hostname, which is unique. For another example, a networked Kubernetes pod has an assigned IPv4 and/or IPv6 address. For another example, a Docker container has an IP address and a hostname.</p> <p><i>See, e.g.:</i></p> <p>VPC requirements and considerations</p> <p>When you create a cluster, the VPC that you specify must meet the following requirements and considerations:</p> <ul style="list-style-type: none"> • The VPC must have a sufficient number of IP addresses available for the cluster, any nodes, and other Kubernetes resources that you want to create. If the VPC that you want to use doesn't have a sufficient number of IP addresses, try to increase the number of available IP addresses. <p>You can do this by updating the cluster configuration to change which subnets and security groups the cluster uses. You can update from the AWS Management Console, the latest version of the AWS CLI, AWS CloudFormation, and eksctl version v0.164.0-rc.0 or later. You might need to do this to provide subnets with more available IP addresses to successfully upgrade a cluster version.</p> <p>https://docs.aws.amazon.com/eks/latest/userguide/network_reqs.html</p> <ul style="list-style-type: none"> • If you want Kubernetes to assign IPv6 addresses to Pods and services, associate an IPv6 CIDR block with your VPC. For more information, see Associate an IPv6 CIDR block with your VPC in the Amazon VPC User Guide. <p>https://docs.aws.amazon.com/eks/latest/userguide/network_reqs.html</p>

Claim 6	Accused Instrumentalities
	<p>Kubernetes control plane software decides when and where to run your pods, manages traffic routing, and scales your pods based on utilization or other metrics that you define. Kubernetes automatically starts pods on your cluster based on their resource requirements and automatically restarts pods if they or the instances they are running on fail. Each pod is given an IP address and a single DNS name, which Kubernetes uses to connect your services with each other and external traffic.</p> <p>https://aws.amazon.com/kubernetes/</p> <h2>Container information</h2> <p>The <i>hostname</i> of a Container is the name of the Pod in which the Container is running. It is available through the <code>hostname</code> command or the <code>gethostname</code> function call in libc.</p> <p>The Pod name and namespace are available as environment variables through the downward API.</p> <p>User defined environment variables from the Pod definition are also available to the Container, as are any environment variables specified statically in the container image.</p> <p>https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 6	Accused Instrumentalities
	<h2 data-bbox="636 245 1234 293">IP address and hostname</h2> <p data-bbox="636 337 1896 461">By default, the container gets an IP address for every Docker network it attaches to. A container receives an IP address out of the IP subnet of the network. The Docker daemon performs dynamic subnetting and IP address allocation for containers. Each network also has a default subnet mask and gateway.</p> <p data-bbox="636 511 1877 683">You can connect a running container to multiple networks, either by passing the <code>--network</code> flag multiple times when creating the container, or using the <code>docker network connect</code> command for already running containers. In both cases, you can use the <code>--ip</code> or <code>--ip6</code> flags to specify the container's IP address on that particular network.</p> <p data-bbox="636 734 1883 857">In the same way, a container's hostname defaults to be the container's ID in Docker. You can override the hostname using <code>--hostname</code>. When connecting to an existing network using <code>docker network connect</code>, you can use the <code>--alias</code> flag to specify an additional network alias for the container on that network.</p> <p data-bbox="621 889 1050 920">https://docs.docker.com/network/</p>

Claim 8

Claim 8	Accused Instrumentalities
<p data-bbox="203 1078 569 1328">8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.</p>	<p data-bbox="621 1078 1911 1182">Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.</p> <p data-bbox="621 1208 1883 1279">For example, AWS EKS, AWS ECS, and AWS E2C can retrieve container images from Amazon's Elastic Container Registry (ECR).</p> <p data-bbox="621 1305 737 1336"><i>See, e.g.:</i></p>

Claim 8	Accused Instrumentalities
	<p>Q: What is Amazon Elastic Container Registry (Amazon ECR)?</p> <p>Amazon ECR is a fully managed container registry that makes it easy for developers to share and deploy container images and artifacts. Amazon ECR is integrated with Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and AWS Lambda, simplifying your development to production workflow. Amazon ECR eliminates the need to operate your own container repositories or worry about scaling the underlying infrastructure. Amazon ECR hosts your images in a highly available and scalable architecture, allowing you to deploy containers for your applications reliably. Integration with AWS Identity and Access Management (IAM) provides resource-level control of each repository that lets you share images across your organization or with anyone in the world.</p> <p>https://aws.amazon.com/ecr/faqs/</p> <p>Q: Why should I use Amazon ECR?</p> <p>Amazon ECR eliminates the need to operate and scale the infrastructure required to power your container registry. Amazon ECR uses Amazon Simple Storage Service (S3) for storage to make your container images highly available and accessible, allowing you to deploy new containers for your applications reliably. Amazon ECR transfers your container images over HTTPS and automatically encrypts your images at rest. You can configure policies to manage permissions for each repository and restrict access to IAM users, roles, or other AWS accounts. Amazon ECR integrates with Amazon ECS, Amazon EKS, AWS Fargate, AWS Lambda, and the Docker CLI, allowing you to simplify your development and production workflows. You can easily push your container images to Amazon ECR using the Docker CLI from your development machine, and Amazon container orchestrators or compute can pull them directly for production deployments.</p> <p>https://aws.amazon.com/ecr/faqs/</p> <p>Q: How do I pull a public image from Amazon ECR?</p> <p>You pull using the familiar 'docker pull' command with the URL of the image. You can easily search for this URL by finding images using a publisher alias, image name, or image description using the Amazon ECR public gallery. Image URLs are in the format public.ecr.aws/<alias>/<image>:<tag>, for example public.ecr.aws/eks/aws-alb-ingress-controller:v1.1.5</p> <p>https://aws.amazon.com/ecr/faqs/</p>

Claim 8	Accused Instrumentalities
	<p>Q: Does Amazon ECR support the Open Container Initiative (OCI) format?</p> <p>Yes. Amazon ECR is compatible with the Open Container Initiative (OCI) image specification, letting you push and pull OCI images and artifacts. Amazon ECR can also translate between Docker Image Manifest V2, Schema 2 images and OCI images on pull.</p> <p>https://aws.amazon.com/ecr/faqs/</p>

Claim 9

Claim 9	Accused Instrumentalities
<p>9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.</p>	<p>Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.</p> <p>For example, Kubernetes tracks and limits resource usage, including CPU and memory resources. For another example, Docker tracks and limits resource usage, including CPU and memory resources.</p> <p><i>See, e.g.:</i></p>

Claim 9	Accused Instrumentalities
	<p data-bbox="640 256 890 297">⇒ Kubecost</p> <p data-bbox="682 349 1885 727">Amazon EKS supports Kubecost, which you can use to monitor your costs broken down by Kubernetes resources including Pods, nodes, namespaces, and labels. As a Kubernetes platform administrator and finance leader, you can use Kubecost to visualize a breakdown of Amazon EKS charges, allocate costs, and charge back organizational units such as application teams. You can provide your internal teams and business units with transparent and accurate cost data based on their actual AWS bill. Moreover, you can also get customized recommendations for cost optimization based on their infrastructure environment and usage patterns within their clusters. For more information about Kubecost, see the Kubecost documentation.</p> <p data-bbox="682 771 1864 846">Amazon EKS provides an AWS optimized bundle of Kubecost for cluster cost visibility. You can use your existing AWS support agreements to obtain support.</p> <p data-bbox="621 878 1535 911">https://docs.aws.amazon.com/eks/latest/userguide/cost-monitoring.html</p> <p data-bbox="716 938 1472 971"><u>Resource Management for Pods and Containers</u></p> <p data-bbox="716 1019 1906 1138">When you specify a <u>Pod</u>, you can optionally specify how much of each resource a <u>container</u> needs. The most common resources to specify are CPU and memory (RAM); there are others.</p> <p data-bbox="716 1187 1906 1349">When you specify the resource <i>request</i> for containers in a Pod, the <u>kube-scheduler</u> uses this information to decide which node to place the Pod on. When you specify a resource <i>limit</i> for a container, the <u>kubelet</u> enforces those limits so that the running container is not allowed to use more of that resource than the</p>

Claim 9	Accused Instrumentalities
	<p>limit you set. The kubelet also reserves at least the <i>request</i> amount of that system resource specifically for that container to use.</p> <p>Requests and limits</p> <p>If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its <i>request</i> for that resource specifies. However, a container is not allowed to use more than its resource <i>limit</i>.</p> <p>For example, if you set a <i>memory</i> request of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the container can try to use more RAM.</p> <p>If you set a <i>memory</i> limit of 4GiB for that container, the kubelet (and <u>container runtime</u>) enforce the limit. The runtime prevents the container from using more than the configured resource limit. For example: when a process in the container tries to consume more than the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an out of memory (OOM) error.</p> <p>Limits can be implemented either reactively (the system intervenes once it sees a violation) or by enforcement (the system prevents the container from ever exceeding the limit). Different runtimes can have different ways to implement the same restrictions.</p> <p>https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</p> <p>Runtime options with Memory, CPUs, and GPUs</p>

Claim 9	Accused Instrumentalities
	<p>By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler allows. Docker provides ways to control how much memory, or CPU a container can use, setting runtime configuration flags of the <code>docker run</code> command. This section provides details on when you should set such limits and the possible implications of setting them.</p> <p>Limit a container's access to memory Docker can enforce hard or soft memory limits.</p> <ul style="list-style-type: none"> • Hard limits lets the container use no more than a fixed amount of memory. • Soft limits lets the container use as much memory as it needs unless certain conditions are met, such as when the kernel detects low memory or contention on the host machine. <p>https://docs.docker.com/config/containers/resource_constraints/</p>

Claim 10

Claim 10	Accused Instrumentalities
<p>10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.</p>	<p>Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.</p> <p><i>See, e.g.:</i></p>

Claim 10	Accused Instrumentalities
	<p data-bbox="642 269 1136 310">What is containerization?</p> <p data-bbox="642 345 1881 505">Containerization is a software deployment process that bundles an application's code with all the files and libraries it needs to run on any infrastructure. Traditionally, to run any application on your computer, you had to install the version that matched your machine's operating system. For example, you needed to install the Windows version of a software package on a Windows machine. However, with containerization, you can create a single software package, or <a data-bbox="1367 448 1455 469" href="https://aws.amazon.com/what-is/containerization/">container, that runs on all types of devices and operating systems.</p> <p data-bbox="621 529 1255 560">https://aws.amazon.com/what-is/containerization/</p> <p data-bbox="642 597 1297 638">How does containerization work?</p> <p data-bbox="642 673 1898 833">Containerization involves building self-sufficient software packages that perform consistently, regardless of the machines they run on. Software developers create and deploy container images—that is, files that contain the necessary information to run a containerized application. Developers use containerization tools to build container images based on the Open Container Initiative (OCI) image specification. OCI is an open-source group that provides a standardized format for creating container images. Container images are read-only and cannot be altered by the computer system.</p> <p data-bbox="642 873 1566 894">Container images are the top layer in a containerized system that consists of the following layers.</p> <p data-bbox="621 915 1255 946">https://aws.amazon.com/what-is/containerization/</p>

Claim 10	Accused Instrumentalities
	<p>Infrastructure</p> <p>Infrastructure is the hardware layer of the container model. It refers to the physical computer or bare-metal server that runs the containerized application.</p> <p>Operating system</p> <p>The second layer of the containerization architecture is the operating system. Linux is a popular operating system for containerization with on-premise computers. In cloud computing, developers use cloud services such as AWS EC2 to run containerized applications.</p> <p>Container engine</p> <p>The container engine, or container runtime, is a software program that creates containers based on the container images. It acts as an intermediary agent between the containers and the operating system, providing and managing resources that the application needs. For example, container engines can manage multiple containers on the same operating system by keeping them independent of the underlying infrastructure and each other.</p> <p>Application and dependencies</p> <p>The topmost layer of the containerization architecture is the application code and the other files it needs to run, such as library dependencies and related configuration files. This layer might also contain a light guest operating system that gets installed over the host operating system.</p> <p>https://aws.amazon.com/what-is/containerization/</p>

Claim 10	Accused Instrumentalities
	<p>How Docker containers work</p> <p>A Docker container is a runtime environment with all the necessary components—like code, dependencies, and libraries—needed to run the application code without using host machine dependencies. This container runtime runs on the engine on a server, machine, or cloud instance. The engine runs multiple containers depending on the underlying resources available.</p> <p>To deploy and scale a set of containers to communicate effectively across different machines or virtual machines, you need a container orchestration platform like Kubernetes. This helps whether your machines are on premises or in the cloud. Kubernetes manages multiple machines, known as a cluster, within the context of container operations.</p> <p>Read about Kubernetes »</p> <p>How Docker images work</p> <p>A Docker image, or container image, is a standalone, executable file used to create a container. This container image contains all the libraries, dependencies, and files that the container needs to run. A Docker image is shareable and portable, so you can deploy the same image in multiple locations at once—much like a software binary file.</p> <p>You can store images in registries to keep track of complex software architectures, projects, business segments, and user group access. For instance, the public Docker Hub registry contains images such as operating systems, programming language frameworks, databases, and code editors.</p> <p>https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/</p>

Claim 10	Accused Instrumentalities
	<p>Container technology uses the resource-isolation features of the Linux kernel to sandbox an application, its dependencies, configuration files, and interfaces inside an atomic unit called a container. This allows a container to run on any host with the suitable kernel components, while shielding the application from behavioral inconsistencies through variances in software installed on the host. Containers use operating system (OS) level virtualization compared to VMs, which use hardware level virtualization using hypervisor. A hypervisor is a software or a firmware that creates and runs VMs. Multiple containers can run on a single host OS without needing a hypervisor, while isolated from neighboring containers. This layer of isolation allows consistency, flexibility, and portability, which enable rapid software deployment and testing. There are many ways in which using containers on AWS can benefit your organization. Containers have been widely employed in use cases such as distributed applications, batch jobs, and continuous deployment pipelines. The use cases for containers continue to grow in areas like distributed data processing, streaming media delivery, genomics, and machine learning, including generative AI.</p> <p>https://docs.aws.amazon.com/whitepapers/latest/containers-on-aws/containers-on-aws.html</p> <p>Before we get too deep into technical details, I want to talk about how containers are typically used and why we see some consistent feedback about those themes. In any environment, booting a computer can take a while. But what's harder than booting is deploying a random application to that computer, and doing so reliably. Containers make this process a lot easier. A container image provides a reliable and repeatable mechanism for packaging up the set of local dependencies for an application, including its dynamically linked libraries, other programs to invoke, and assets. The Linux kernel primitives that power containers, including cgroups and namespaces, provide some amount of resource and visibility isolation. Containers also start up much more quickly than a whole computer. These properties enable each application to pretend that it's the only application running, enables subdividing larger computers into smaller parts so more of these applications can run together without conflict, and makes it attractive to use one computer for running multiple applications or even a cluster of computers to run many copies of those applications.</p> <p>https://aws.amazon.com/blogs/containers/bottlerocket-a-special-purpose-container-operating-system/</p>

Claim 10	Accused Instrumentalities
	<p data-bbox="653 266 1314 318">Build secure microservices</p> <p data-bbox="653 362 1881 643">Ensure strong security isolation between your containers. AWS provides the latest security updates and lets you set granular access permissions for every container. AWS offers over 210 security, compliance, and governance services, plus key features to best suit your needs.</p> <p data-bbox="621 680 1077 711">https://aws.amazon.com/containers/</p> <p data-bbox="642 748 856 789">Understand</p> <p data-bbox="642 829 1661 857">Containers offer a number of advantages for packaging, deploying, and running applications:</p> <ul data-bbox="653 889 1871 956" style="list-style-type: none">• Isolation: Improve security and reliability with containers' process-level isolation, with which applications running in separate containers cannot interfere with each other, improving security and reliability. <p data-bbox="621 974 1917 1040">https://docs.aws.amazon.com/decision-guides/latest/containers-on-aws-how-to-choose/choosing-aws-container-service.html</p>

Claim 10	Accused Instrumentalities
	<p data-bbox="638 248 810 272">Fault tolerance</p> <p data-bbox="638 313 1877 402">Software development teams use containers to build fault-tolerant applications. They use multiple containers to run microservices on the cloud. Because containerized microservices operate in isolated user spaces, a single faulty container doesn't affect the other containers. This increases the resilience and availability of the application.</p> <p data-bbox="638 440 716 464">Agility</p> <p data-bbox="638 500 1877 589">Containerized applications run in isolated computing environments. Software developers can troubleshoot and change the application code without interfering with the operating system, hardware, or other application services. They can shorten software release cycles and work on updates quickly with the container model.</p> <p data-bbox="621 630 1255 662">https://aws.amazon.com/what-is/containerization/</p> <p data-bbox="638 703 1898 776">Linux containers are made up of control groups (cgroups) and namespaces that help limit what a container can access, but all containers share the same Linux kernel as the host Amazon EC2 instance.</p> <p data-bbox="621 792 1430 824">https://docs.aws.amazon.com/eks/latest/userguide/security.html</p>

Claim 10	Accused Instrumentalities
	<p>Use Windows containers instead of running many applications on one instance of IIS</p> <p>Consider the following advantages of using Windows containers instead of running multiple applications on one EC2 Windows instance with Internet Information Services (IIS):</p> <ul style="list-style-type: none"> • Security – Containers provide a level of security out of the box that isn't achieved through isolation at the IIS level. If one IIS website or application is compromised, all the other hosted sites are exposed and vulnerable. Container escape is rare and a harder vulnerability to exploit than gaining control of a server through a web vulnerability. • Flexibility – The ability to run containers in process isolation and have their own instance allows for more granular networking options. Containers also offer complex distribution methods across many EC2 instances. You don't get these benefits when you consolidate applications on a single IIS instance. • Management overhead – Server Name Indication (SNI) creates overhead that requires management and automation. Also, you have to grapple with typical operating system management operations like patching, troubleshooting BSOD (if auto scaling isn't in place), endpoint protection, and so on. Configuring IIS sites according to security best practices is a time consuming and ongoing activity. You might even need to set up trust levels, which also adds to management overhead. Containers are designed to be stateless and immutable. Ultimately, your deployments are faster, more secure, and repeatable if you use Windows containers instead. <p>https://docs.aws.amazon.com/prescriptive-guidance/latest/optimize-costs-microsoft-workloads/windows-containers-main.html</p>

Claim 13

Claim 13	Accused Instrumentalities
13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when	Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

Claim 13	Accused Instrumentalities
processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.	<i>See</i> analysis and evidence for claim 1 above. <i>See also, e.g.:</i>

Claim 13	Accused Instrumentalities
	<h2 data-bbox="653 256 1499 375">Logging and Monitoring in Amazon Elastic Container Service</h2> <p data-bbox="653 407 760 431"> PDF RSS </p> <p data-bbox="653 488 1661 662">Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Elastic Container Service and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon ECS resources and responding to potential incidents:</p> <p data-bbox="653 699 978 724">Amazon CloudWatch Alarms</p> <p data-bbox="680 740 1608 987">Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Monitor Amazon ECS using CloudWatch .</p> <p data-bbox="680 1024 1633 1157">For services with tasks that use the Fargate launch type, you can use CloudWatch alarms to scale in and scale out the tasks in your service based on CloudWatch metrics, such as CPU and memory utilization. For more information, see Automatically scale your Amazon ECS service.</p> <p data-bbox="680 1195 1629 1292">For clusters with tasks or services using the EC2 launch type, you can use CloudWatch alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation.</p> <p data-bbox="621 1308 1829 1341"> https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-logging-monitoring.html </p>

Claim 13	Accused Instrumentalities
	<p>Amazon CloudWatch Logs</p> <p>Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the <code>awslogs</code> log driver in your task definitions. For more information, see Using the awslogs driver.</p> <p>You can also monitor, store, and access the operating system and Amazon ECS container agent log files from your Amazon ECS container instances. This method for accessing logs can be used for containers using the EC2 launch type..</p> <p>Amazon CloudWatch Events</p> <p>Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see Automate responses to Amazon ECS errors using EventBridge in this guide and What Is Amazon CloudWatch Events? in the <i>Amazon CloudWatch Events User Guide</i>.</p> <p>AWS CloudTrail Logs</p> <p>CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon ECS. Using the information collected by CloudTrail, you can determine the request that was made to Amazon ECS, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Log Amazon ECS API calls using AWS CloudTrail.</p> <p>https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-logging-monitoring.html</p>

Claim 13	Accused Instrumentalities
	<p>AWS Compute Optimizer</p> <p>AWS Compute Optimizer is a service that analyzes the configuration and utilization metrics of your AWS resources. It reports whether your resources are optimal, and generates optimization recommendations to reduce the cost and improve the performance of your workloads.</p> <p>For more information, see AWS Compute Optimizer recommendations for Amazon ECS.</p> <p>Another important part of monitoring Amazon ECS involves manually monitoring those items that the CloudWatch alarms don't cover. The CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your container instances and the containers in your tasks.</p> <p>https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-logging-monitoring.html</p>

Claim 14

Claim 14	Accused Instrumentalities
<p>14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:</p> <p>a) running an instance of a service on a server;</p>	<p>Amazon and/or its customer practices, through the Accused Instrumentalities, a method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by (a) running an instance of a service on a server; (b) determining which files are being used; and, (c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.</p> <p>For example, AWS App2Container, used in conjunction with ECS, EKS, and/or EC2, supports the creation of containers and deploying the containers on the server(s). The containers are first created</p>

Claim 14	Accused Instrumentalities
<p>b) determining which files are being used; and,</p> <p>c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.</p>	<p>and then later deployed/stored on the server. The creation step involves determining which applications and files are to be migrated, copying these identified applications and files to a location in the target server. Based on information and belief, once the files are migrated, the earlier stored files (if any) are not deleted/overwritten, rather, the migrated files are stored as different instance in memory accessible to containers. Further, an instance of an application/service may be tested or run on the target server to ensure compatibility.</p> <p><i>See analysis and evidence for claims 1 and 4 above.</i></p> <p><i>See also, e.g.:</i></p> <p>Q: What is AWS App2Container ?</p> <p>AWS App2Container (A2C) is a new command line tool that helps transform existing applications running in virtual machines into containers, without needing any code changes. A2C discovers applications running on a server, identifies dependencies, and generates relevant artifacts for seamless deployment to Amazon ECS and EKS. A2C also provides integration with AWS CodeBuild and CodeDeploy to enable a repeatable way to build and deploy containerized application.</p> <p>https://aws.amazon.com/app2container/faqs/</p>

Claim 14	Accused Instrumentalities
	<p>Q: How does App2Container make it easier to containerize applications?</p> <p>App2Container enables you to quickly transform your existing applications running on-premises, in Amazon EC2, or in any other cloud. You get the following benefits by using A2C for containerization:</p> <ul style="list-style-type: none"> • Application Inventory: A2C identifies the supported ASP.NET and Java applications running in a server which enables a quick and accurate inventory of applications in your environment without extensive manual effort. • Detailed Dependency Analysis: A2C analyzes the running application and identifies dependencies including cooperating processes and network port dependencies. This detailed information reduces the manual effort required to understand and document application anatomy and required dependencies. • Seamless Deployment: A2C generates ECS task definitions and Kubernetes deployment YAML for the containerized application following the AWS best practices for security and scalability by integrating with various AWS services such as ECR, ECS, and EKS. A2C generates CloudFormation template to configure required compute, network, and security infrastructure to seamlessly deploy containerized application in AWS. A2C creates CI/CD pipelines for Amazon DevOps services such as CodeBuild and CodeDeploy to build and deploy containers. If you have existing CI/CD tooling (for example, Azure DevOps, and Jenkins), then you can integrate A2C provided deployment artifacts into your existing CI/CD workflows. <p>https://aws.amazon.com/app2container/faqs/</p>

Claim 14	Accused Instrumentalities
	<p>Q: How do I get started?</p> <p>Once the prerequisites are met and the App2Container tool is installed on the application server, use the following sequence of commands to containerize the application.</p> <ul style="list-style-type: none"> • Init: Configures the tool with your AWS profile and other related parameters. • Inventory: Identifies all running applications that are supported by A2C. Choose an application to proceed. • Analyze: Performs detailed analysis to identify application artifacts, third-party dependencies, network ports and configuration files, and generates a configuration file to represent the gathered information. You can make additional changes to the configuration file (for example, exclude or include any additional files or directories). • Containerize: Generates a dockerfile and container image for the analyzed application. If needed, you can modify the A2C-generated dockerfile and rebuild the container. You can also perform local testing of the containerized application using docker run command before deploying to AWS container services. • Generate Deployment: Registers the container image with Amazon ECR, creates ECS task definitions, and Kubernetes deployment YAML file to deploy the containerized application to Amazon ECS or Amazon EKS. A2C generates CloudFormation template to create required infrastructure to deploy the containerized application. Optionally, it also generates AWS CodeBuild and CodeDeploy artifacts to enable repeatable way to build and deploy the containerized app. <p>https://aws.amazon.com/app2container/faqs/</p> <p>Q: What artifacts are generated by App2Container?</p> <p>App2Container generates the following artifacts for each application component: 1) Application artifacts such as DLLs configuration, 2) Dockerfile, 3) container image in ECR, 4) ECS Task definitions, 5) Kubernetes deployment YAML, 6) CloudFormation Template, 6) AWS CodeBuild and CodeDeploy resources.</p> <p>https://aws.amazon.com/app2container/faqs/</p>